



# **NAVAL POSTGRADUATE SCHOOL**

**MONTEREY, CALIFORNIA**

## **THESIS**

**MULTI-TARGET TRACKING FOR SWARM VS. SWARM  
UAV SYSTEMS**

by

Umit Soylu

September 2012

Thesis Advisor:  
Second Reader:

Timothy H. Chung  
Joel Young

**Approved for public release; distribution is unlimited**

THIS PAGE INTENTIONALLY LEFT BLANK

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

<b>1. REPORT DATE</b> (DD-MM-YYYY) 21-09-2012		<b>2. REPORT TYPE</b> Master's Thesis		<b>3. DATES COVERED</b> (From — To) 2010-08-10—2012-08-21	
<b>4. TITLE AND SUBTITLE</b>  Multi-Target Tracking for Swarm vs. Swarm UAV Systems				<b>5a. CONTRACT NUMBER</b>	
				<b>5b. GRANT NUMBER</b>	
				<b>5c. PROGRAM ELEMENT NUMBER</b>	
<b>6. AUTHOR(S)</b>  Umit Soylu				<b>5d. PROJECT NUMBER</b>	
				<b>5e. TASK NUMBER</b>	
				<b>5f. WORK UNIT NUMBER</b>	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b>  Naval Postgraduate School Monterey, CA 93943				<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b>  Department of the Navy				<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b>	
				<b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b>	
<b>12. DISTRIBUTION / AVAILABILITY STATEMENT</b>  Approved for public release; distribution is unlimited					
<b>13. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A					
<b>14. ABSTRACT</b>  Unmanned systems, including unmanned aerial vehicles (UAVs), are developing technologies that are becoming increasingly important. This thesis provides a model for generating a common operational picture (COP) for unmanned systems that is applicable in today's technology, and presents results and analysis based on simulation studies. This thesis specifically investigates a swarm versus swarm unmanned systems scenario in which opposing teams of UAVs approach each other. Different methodologies for generating a COP from the perspective of a given team are investigated, and a simulation is designed to explore the performance of the selected strategies for performing multi-target tracking. The results of the simulation show the performance of the presented approach where targets are assumed in the field of view of the tracking agents, false detections may or may not be present, and all entities maneuver according to nondeterministic motion models.					
<b>15. SUBJECT TERMS</b> UAV, Unmanned systems, Swarm vs. Swarm systems, Agent-based simulations, Common Operational Picture, K-means clustering, L-method, Polynomial Fit, Munkres algorithm, Augmented Suboptimal Joint Probabilistic Data Association, Kalman Filtering					
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT</b>  UU	<b>18. NUMBER OF PAGES</b>  119	<b>19a. NAME OF RESPONSIBLE PERSON</b>
<b>a. REPORT</b> Unclassified	<b>b. ABSTRACT</b> Unclassified	<b>c. THIS PAGE</b> Unclassified			<b>19b. TELEPHONE NUMBER</b> (include area code)

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release; distribution is unlimited**

**MULTI-TARGET TRACKING FOR SWARM VS. SWARM UAV SYSTEMS**

Umit Soylu  
Lieutenant Junior Grade, Turkish Navy  
B.S., Turkish Naval Academy, 2007

Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL  
September 2012**

Author: Umit Soylu

Approved by: Timothy H. Chung  
Thesis Advisor

Joel Young  
Second Reader

Peter J. Denning  
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

## **ABSTRACT**

Unmanned systems, including unmanned aerial vehicles (UAVs), are developing technologies that are becoming increasingly important. This thesis provides a model for generating a common operational picture (COP) for unmanned systems that is applicable in today's technology, and presents results and analysis based on simulation studies. This thesis specifically investigates a swarm versus swarm unmanned systems scenario in which opposing teams of UAVs approach each other. Different methodologies for generating a COP from the perspective of a given team are investigated, and a simulation is designed to explore the performance of the selected strategies for performing multi-target tracking. The results of the simulation show the performance of the presented approach where targets are assumed in the field of view of the tracking agents, false detections may or may not be present, and all entities maneuver according to nondeterministic motion models.

THIS PAGE INTENTIONALLY LEFT BLANK

---

---

# Table of Contents

---

<b>List of Acronyms and Abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Related Works . . . . .	2
1.3 Main Contributions of the Thesis . . . . .	5
1.4 Organization . . . . .	6
1.5 Limitations and Assumptions . . . . .	6
<b>2 Model Formulation</b>	<b>9</b>
2.1 Simulation Implementation . . . . .	9
2.2 Agent Modeling. . . . .	16
2.3 Environment Modeling . . . . .	18
<b>3 Distributed Target Tracking from Multiple Sensors</b>	<b>21</b>
3.1 General Flow . . . . .	21
3.2 Detection Processing . . . . .	24
3.3 Track Generation . . . . .	27
3.4 Track Association . . . . .	44
3.5 Kalman Filtering . . . . .	57
<b>4 Simulation Results</b>	<b>61</b>
4.1 General Outline . . . . .	61
4.2 Scenario One – Baseline . . . . .	62
4.3 Scenario Two – Many Sensors, Many Targets . . . . .	68

4.4	Scenario Three – Impact of False Detections . . . . .	75
4.5	Scenario Four – Sensitivity to False Detections . . . . .	80
<b>5</b>	<b>Conclusion and Future Work</b>	<b>87</b>
5.1	Conclusion. . . . .	87
5.2	Future Work . . . . .	90
	<b>Initial Distribution List</b>	<b>97</b>

---



---

## List of Figures

---

Figure 2.1	Illustrations of the motivating swarm vs. swarm scenario: (a) represents different flight phases for defending UAVs, (b) represents a generic swarm vs. swarm system, and (c) represents a swarm attack and swarm defense scenario for a high valued unit. . . . .	10
Figure 2.2	Spherical coordinates description with Cartesian coordinates. . . . .	12
Figure 2.3	An illustration of geometric occlusion in generating detections. The red and blue agents (right) block portions of the field of view of the black (left) agent. . . . .	14
Figure 2.4	An example of spherical rotation. The yellow circle (positioned lower middle) represents the original vector and red circle (positioned upper middle) represents the rotated vector with angles $\theta$ and $\phi$ . . . . .	16
Figure 3.1	General information flow of an agent. State, Detections, and NetworkDetections are generated by the simulation and provided as input to the agent. This figure depicts the model tested for any discrete time step. .	22
Figure 3.2	A sample graph for number of clusters against total distance to associated centroids for all observation. . . . .	30
Figure 3.3	An example graph for polynomial fit. It should be noted that the graph consists of $\log(x)$ and $y$ parameters of the data in Section 3.3.1.1. . . .	33
Figure 3.4	Possibilities of line fits for seven data points (from [13]) . . . . .	34
Figure 3.5	Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. RealValue shows the exact number of tracks, ‘real k’, that originates the detections. Methods that finds closer results to ‘real k’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. . . . .	37

Figure 3.6	Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. RealValue shows the exact number of tracks, ‘real k’, that originates the detections. Methods that finds closer results to ‘real k’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. . . . .	38
Figure 3.7	Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. ‘SumDistanceToCentroid’ shows the exact curve of the data. Methods that find closer results to ‘# of Tracks’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. The location of the index boxes that shows the ‘Best-K’ the associated method is found, is slightly shifted for readability purposes. . . . .	39
Figure 3.8	Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. ‘SumDistanceToCentroid’ shows the exact curve of the data. Methods that find closer results to ‘# of Tracks’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. The location of the index boxes that shows the ‘Best-K’ the associated method is found, is slightly shifted for readability purposes. . . . .	40
Figure 3.9	PDF distribution of vector $x$ with $\chi = 0$ , $\Sigma = \sigma$ where the probability of a point belongs to the vector $x$ is computed as the area under the curve. (Figure taken from [35].) . . . . .	52
Figure 4.1	$x$ dimension views of Target 1 and Agent 1’s closest track in Scenario One. (a) The $x$ dimension plots for both Target 1 and Agent 1’s closest track (b) The $x$ dimension error of Agent 1’s track and Target 1 . . . .	64
Figure 4.2	$y$ dimension views of Target 1 and Agent 1’s closest track in Scenario One. (a) The $y$ dimension plots for both Target 1 and Agent 1’s closest track (b) The $y$ dimension error of Agent 1’s track and Target 1 . . . .	65
Figure 4.3	$z$ dimension views of Target 1 and Agent 1’s closest track in Scenario One. (a) The $z$ dimension plots for both Target 1 and Agent 1’s closest track (b) The $z$ dimension error of Agent 1’s track and Target 1 . . . .	66

Figure 4.4	Box plots for the tracking error for all state variables of the agents over all tracks and associated target pairings in Scenario One. Average values show exponential convergence to steady state, with large covariances (from newly instantiated tracks) diminishing quickly upon continuous tracking. . . . .	67
Figure 4.5	The mean number of associated tracks at time $t$ vs. tracks at time $t - 1$ for all agents in Scenario One. If tracks have no association, they are accepted as new tracks without prior knowledge. . . . .	68
Figure 4.6	An example configuration of simulation at time $t = 1$ of Scenario Two. Red markers (diamonds) are agents, blue markers (circles) are targets and green marker (hexagram) is a track. . . . .	69
Figure 4.7	The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Two. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of agents and tracks. This figure exemplifies the trace of agents with tracks. . . .	70
Figure 4.8	$x$ dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The $x$ dimension plots for both Target 1 and Agent 1's closest track (b) The $x$ dimension error of Agent 1's track and Target 1 . . . .	71
Figure 4.9	$y$ dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The $y$ dimension plots for both Target 1 and Agent 1's closest track (b) The $y$ dimension error of Agent 1's track and Target 1 . . . .	71
Figure 4.10	$z$ dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The $z$ dimension plots for both Target 1 and Agent 1's closest track (b) The $z$ dimension error of Agent 1's track and Target 1 . . . .	72
Figure 4.11	Box plots for the tracking error for all state variables of the agents over all tracks and associated target pairings in Scenario Two. Throughout the scenario, there exists an average tracking errors for each dimension that results due to the randomization of all agents. . . . .	73
Figure 4.12	The mean number associated tracks at time $t$ vs. tracks at time $t - 1$ for all agents in Scenario Two. If tracks have no association, they are accepted as new tracks without prior knowledge . . . . .	74
Figure 4.13	$x$ dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The $x$ dimension plots for both Target 1 and Agent 1's closest track (b) The $x$ dimension error of Agent 1's track and Target 1 . . . .	75

Figure 4.14	y dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The y dimension plots for both Target 1 and Agent 1's closest track (b) The y dimension error of Agent 1's track and Target 1 . . . . .	76
Figure 4.15	z dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The z dimension plots for both Target 1 and Agent 1's closest track (b) The z dimension error of Agent 1's track and Target 1 . . . . .	76
Figure 4.16	The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Three. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of <i>agents</i> and tracks. This figure exemplifies the trace of agents with tracks . . . .	77
Figure 4.17	The mean number associated tracks at time $t$ vs. tracks at time $t - 1$ for all agents in Scenario Three. If tracks have no association, they are accepted as new tracks without prior knowledge . . . . .	77
Figure 4.18	This figure, in Scenario Three, depicts the mean error of all agents for each dimension in terms of the difference of each track and its associated target. . . . .	79
Figure 4.19	x dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The x dimension plots for both Target 1 and Agent 1's closest track (b) The x dimension error of Agent 1's track and Target 1 . . . . .	81
Figure 4.20	y dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The y dimension plots for both Target 1 and Agent 1's closest track (b) The y dimension error of Agent 1's track and Target 1 . . . . .	81
Figure 4.21	z dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The z dimension plots for both Target 1 and Agent 1's closest track (b) The z dimension error of Agent 1's track and Target 1 . . . . .	82
Figure 4.22	The mean number associated tracks at time $t$ vs. tracks at time $t - 1$ for all agents in Scenario Four. If tracks have no association, they are accepted as new tracks without prior knowledge . . . . .	83
Figure 4.23	The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Four. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of agents and tracks. This figure exemplifies the trace of agents with tracks . . . .	84
Figure 4.24	This figure, in Scenario Four, depicts the mean error of all agents for each dimension in terms of the difference of each track and its associated target. . . . .	85

---



---

## List of Tables

---

Table 3.1	Table of notations and definitions, including relevant vector or matrix dimensions . . . . .	24
Table 3.2	Table of notations and definitions, including relevant vector or matrix dimensions . . . . .	25
Table 3.3	Table of notations and definitions, including relevant vector or matrix dimensions . . . . .	26
Table 3.4	For observations $\in \mathbb{R}^3$ with total of 4 observation, $k$ is selected starting from 2 . . . 4. In this table, $k = 2$ . Each observation is associated with one of the centroids. Total distance in $k = 2$ is $\sum Distance = 6.92$ . . . . .	30
Table 3.5	The results of selecting best K algorithms. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3, and LogExpFit is described in Section 3.3.1.2. The red numbers represent the closest estimate in terms of the real number of tracks. . . . .	41
Table 3.6	For each scenario, the parameters of the scenario are described under the ‘Real Parameters’ column, and the distance from the $x$ dimension of the ‘real index’, or ‘Best- $K$ ’, or ‘Number of Tracks’, is described under the ‘Height of the index’ column. The real index distance is normalized for each scenario. The methods described in Section 3.3.1.1 and Section 3.3.1.2 are using %0.9 reduction ratio from the maximum distance which is same as 0.1000 ‘Height of the index’ in this table. The normalization is done for each scenario itself. It should be noted that normalization is not done after all heights for all scenarios are computed. Each normalization of ‘Height of the index’ is independent from any other scenario. . . . .	41
Table 4.1	Simulation Parameters . . . . .	63

Table 4.2	The initial states of three agents and three targets in Scenario One. Note that the motion of these agents is deterministic. . . . .	64
Table 5.1	Summary of the average state-estimate errors for each of six state variables (positions and linear speeds) from simulation studies for four different scenarios, described in Section 4. . . . .	88
Table 5.2	Summary of average number of track computation for each scenario. It should be noted that number of targets is fixed throughout the simulation. . . . .	88

---

## List of Acronyms and Abbreviations

---

<b>ADCP</b>	Acoustic Current Doppler Profiler
<b>ASJPDA</b>	Augmented Suboptimal Joint Probabilistic Data Association
<b>CI</b>	Covariance Intersection
<b>CMOMMT</b>	Cooperative Multi-robot Observation of Multiple Moving Targets
<b>COP</b>	Common Operational Picture
<b>FN</b>	False Negative
<b>FOV</b>	Field of View
<b>FP</b>	False Positive
<b>IPPF</b>	Independent Partition Particle Filter
<b>JPDA</b>	Joint Probabilistic Data Association
<b>JPDAF</b>	Joint Probabilistic Data Association Filtering
<b>KF</b>	Kalman Filtering
<b>MAP</b>	Maximum A Posteriori
<b>MC-JPDA</b>	Monte Carlo Joint Probabilistic Data Association Filtering
<b>MSJPDA</b>	Multi Sensor Joint Probabilistic Data Association
<b>MTT</b>	Multi-Target Tracking
<b>NPS</b>	Naval Postgraduate School
<b>PDA</b>	Probabilistic Data Association
<b>PDF</b>	Probability Density Function
<b>RMSE</b>	Root Mean Square Error
<b>SLAM</b>	Simultaneous Localization and Mapping
<b>SSPF</b>	Sequential Sampling Particle Filter
<b>UAV</b>	Unmanned Air Vehicle
<b>WSN</b>	Wireless Sensor Networks

THIS PAGE INTENTIONALLY LEFT BLANK

---

# Executive Summary

---

Unmanned systems, including unmanned aerial vehicles (UAVs), are increasingly critical developing technologies. The advantage of preventing human casualties makes unmanned systems demanding technologies of near future. Given the development of unmanned systems worldwide, swarm vs. swarm UAV conflicts are probable near future scenario. For swarms to succeed, the common operational picture (COP) of each members must be accurate.

This paper proposes methods for generating a COP for each member of the swarm. There exists different methodologies applicable to different parts of the problem. These methodologies are evaluated according to the accuracy of the generated COP for each agent.

A simulation is generated for testing realistic scenarios and providing statistical analysis of COP accuracy. The simulation is capable of generating swarm vs. swarm systems and generating statistics for each UAV in the swarm. In this paper, we assume that UAVs are in the air, have knowledge of opposing force members and can share their knowledge with swarm members via networking.

The simulation generates detections according to the targets in the environments and uses Gaussian and uniform distributions. Both occlusions of the agents and possibilities for false detections are implemented.

The simulation is flexible and allows different scenarios with different parameter sets. The affects of false detections are investigated. There exists predetermined simulation borders and agents bounce back from these borders. Also, agents move randomly.

The results shows the efficiency and drawbacks of different methodologies applied. The future works section discuss possible improvements for generating more accurate COP, associating targets in discrete time steps, and factoring network constraints into COP generation.

THIS PAGE INTENTIONALLY LEFT BLANK

---

---

## Acknowledgements

---

I must mention the support and the knowledge of all my the professors since my arrival to Naval Postgraduate School. Without their support, I could never have manage to finalize my thesis satisfactorily.

My thesis advisor, Timothy Chung, deserves special recognition. His support, knowledge and confidence in me transformed this process into an illuminating and delightful experience. My second reader, Joel Young, provided key insights for the problems I came across and helped me deal with them.

My parents, who live in Turkey, never withheld their support and belief in me. Even from thousands of miles away, I always felt their confidence in me.

Also, I cannot forget my cousin, Selcuk Ozmumcu, co-founder of OZMUMCU<sup>TM</sup> Creative Solutions Ltd., for his support and artistic designs in my thesis. Without his professional-quality graphics, it would have been impossible for me to relay this information with such ease.

THIS PAGE INTENTIONALLY LEFT BLANK

---

# CHAPTER 1:

## Introduction

---

### 1.1 Motivation

Multiple target tracking, investigated since the 1950s, has a wide variety of applications, including radar-based air traffic control, sonar-based marine life detection, vehicle tracking for surveillance systems, and visual tracking using computer vision. Objects of interest for tracking can be equally diverse, and their characteristics depend on the specific application. Of interest is a robust framework for multi-target tracking (MTT) algorithms that can be applied to any of the scenarios described above. The main goal for MTT is to generate an accurate global common operational picture (COP) of the evolution of the system, based on observations from one or many noisy sources.

In particular, tracking multiple targets via distributed sensors, such as those that might be located on a team of cooperating unmanned air vehicles (UAVs), is a promising application adaptable to a wide variety of situations including military reconnaissance missions, public safety missions, and exploring unknown areas. The determination of a global COP with a decentralized fusion algorithm based on the sensor data acquired from this team of UAVs and a scalable approach for conducting multi-target tracking are some of the main challenges.

The key research issues for MTT are determining the accuracy of sensor measurements, improving estimates through filtering, and tracking associations in real time; all within constrained computational resources. This thesis focuses on tracking the evolution of target state estimates from multiple mobile sensors, integrating numerous components to accomplish this task. Sensor measurements are assumed imperfect, that is, perturbed by noise, leading to imperfect data associations and potentially degraded target state estimates. Due to the motion of the sensors themselves, the geometry of their configuration is also considered, including the effects of occlusion of measurements. We ignore communication constraints, such as latency and dropped communications, as well as issues with imperfect localization of the mobile sensors.

The main focus in this paper is generating a common operational picture for each agent and associating tracks in real time. Each agent shares their own knowledge about the environment and generates COP both with its own measurements and with other agents measurements. Also, each agent associates tracks individually at each time step. A single consistent global COP is

not built as part of this thesis.

The multi-target tracking framework presented includes the following key algorithmic components, including: (1) track generation using K-means clustering of detection measurements of targets; (2) track association over time using probabilistic data association (PDA) approaches with the Munkres method; (3) state improvement of tracks with Kalman filtering (KF).

## **1.2 Related Works**

### **1.2.1 Motion Model**

Despite the power of KF, which works recursively and produces estimation of unknown variables from noisy measurements over time, MTT is still a difficult problem, even for a centralized systems. Tracking multiple targets in a distributed-sensor network is investigated by [1], which provides a near-optimal solution with low overhead and low communication. In order to reduce the power consumption, the notion of sleep and wake periods for sensors while managing minimum number of sensors with acceptable sensing quality is mentioned in [1].

MTT by multi-sensors has its own applications such as search/rescue tasks and observation of items in a warehouse. The art gallery problem, the problem of observing whole museum with the minimum number of guards, is a typical problem to which MTT applied. [2] investigates this problem using mobile autonomous agents to observe an unknown and dynamic environment. It investigates algorithms for cooperative, multi-robot observation of multiple moving targets (CMOMMT) and introduces the A-CMOMMT algorithm and compares it with other algorithms to describe its constraints and advantages/disadvantages. Both real-world and detailed simulation results are provided. The author concluded that A-CMOMMT is good for hard problems where the number of targets is greater than the number of agents.

Simulation is the easiest way to test MTT algorithms. Better simulations and detailed tests provide accurate results for the efficiency, accuracy and computational workload of an algorithm according to given criteria. It can be inferred that, for any MTT simulation, a random-walk model is used for agents. A random-walk model is investigated by [3]. Random and independent node motions implemented in [3] and conditions for the existence of a stationary regime, which needs to be averted, are shown. The paper shows that node distribution converges to a time-stationary distribution on convex and non-convex spaces.

Random movement for very large models is different from a random-walk model. This issue

is presented in [4]. In very large models, the random-walk model cannot be achieved unless some selection strategy is applied. [4] introduces interactive transition systems, called ‘reactive modules’, in order to provide a random walk model for very large systems.

For accurate simulations, random path generation in very large models is another important factor investigated by [5]. In [5], it is assumed that very large models are made up of several concurrent components, and the paper tries to combine these component paths such that they results in a uniform drawing of paths in a global system. A discrete-time uniform random walk model is another method that can be used in simulations. Such a model and its properties, such as exit probability over an edge of the simulation and the exit time (to cross across the border), is analyzed by [6].

### **1.2.2 Data Fusion and Estimation**

In MTT, providing nearly accurate state estimates is crucial. One of the well known solutions is KF, which is introduced in [7]. KF is a recursive solution for discrete data linear fitting. It is an optimal estimator under special assumptions, and can be used in a wide variety of situations.

In MTT applications with distributed systems, data fusion, the process of integrating multiple data into a consistent and useful representation, is inevitable. Since the state estimates of targets are acquired via sensor and with the errors of these sensors, the fused error needs to be calculated. Covariance Intersection (CI), further investigated by [8–10], is one such method. For CI, [8] focuses on unknown correlation between the data while [9] tries to provide a solution for simultaneous localization and mapping (SLAM). Also, [10] introduces a fast CI algorithm for unknown correlation between data points.

Data fusion in distributed MTT heavily relies on clustering. For distributed systems where the same target may cause multiple observations, tracks, via multiple sensors, it is important to know which track is originated from the same target. One of the well known methods for this purpose is K-means clustering. A key parameter in K-means clustering is the number of clusters to use. Much research has focused [11–13] on determining the best  $K$  for specific or general purposes.

Reference [14] describes architectures for distributed data-fusion of multiple sensors for tracking and estimating the state and dynamics of a target. It describes advantages of using distributed data-fusion architectures for both linear and non-linear systems with independent measurement errors. Different kinds of distributed data-fusion methods are analyzed with each

algorithm's alignment, association and updating cycle. It also describes the communication requirements for distributed data fusion.

### 1.2.3 Data Association

The correct identification of  $n$  targets with  $m$  tracks throughout time is a hard problem. Even with perfect detection, where  $n = m$ , there exist  $n^m$  different identifications. A global estimator from local track estimations with a local estimation model is described by [15]. [16] introduces a tree weighted approach for converting data association problem to a maximum a posteriori probability configuration (MAP) in a graphical model. Also, accomplishing data association with local message passing for graphical models is investigated by [17]. [18] introduces joint probabilistic data association filtering (JPDAF) methods for data association in SLAM and provides test results of JPDAF. Sample based joint probabilistic data-association (JPDA) is investigated by [19] while topic intensity JPDA is investigated by [20]. Also, [21] introduces a suboptimal method for JPDA with performance comparison of other JPDAF methods.

Reference [22] investigates the performance of sequential and parallel implementation of multi-sensor joint probabilistic data association (MSJPDA) based on simulations. It describes the pros and cons of both parallel and sequential MSJPDA algorithms and provides a brief insight on why the sequential algorithm performs better than a parallel MSJPDA algorithm in a given setup.

### 1.2.4 History

Reference [23] discusses issues related to accurate detections of moving/stable targets with single/multiple UAVs. It compares different approaches for generating accurate detections and provides results based on simulations consisting of targets moving faster/slower than UAVs, and stable/moving targets with/without the presence of wind. It compares all approaches with the same environmental setups and makes inferences according to the results based on simulations.

Associating sensor measurements with target tracks is a challenge in MTT systems. Also, the data association problem in closely stationed targets with an exponentially increased dimensionality due to the state-space associated of multiple targets is a crucial problems that needs to be dealt with. Monte Carlo joint probabilistic data-association filtering (MC-JPDAF), investigated by [24], provides an efficient solution for this problem. It provides a sequential sampling particle filter (SSPF), which samples individual targets sequentially with factorization of the importance of weights, and an independent partition particle filter (IPPF), which assumes asso-

ciations are independent. MC-JPDA is designed for a known number of targets with nonlinear and non-Gaussian target dynamics whereas JPDA is designed for linear and Gaussian target dynamics.

Reference [25] describes a wide variety of MTT algorithms according to their history/batch length, scalability of each algorithm according to dimensionality and number of targets, computational complexity, and the main approach or strategy of each algorithm. The paper states that true/false negative detections are misleading statistics; instead, it introduces track initiation and maintenance of a track as criteria for the MTT algorithm-comparison factor. In [25], all MTT simulations and real world tests are done without knowing the exact number of targets in the environment. The paper introduces suboptimal algorithms for linear systems that can deal with noisy and cluttered environments where false alarms exist. It summarizes a wide variety of MTT algorithms and depicts the results based on both simulation and real-world data.

Clearly MTT is a well-studied topic. [26] provides a survey about MTT algorithms with a comprehensive review of tracking maneuvering targets, including 2D and 3D maneuvering models. Mathematical models used for tracking maneuvering targets are presented in [26]. Maneuvering and non-maneuvering targets (dynamic models and algorithms used for these models) are analyzed, and their pros and cons are represented in the paper.

MTT has a wide variety of applications. Example applications are radar-based tracking of aircraft, sonar-based tracking of sea animals and submarines, video-based tracking of people for security or surveillance, and so on. Missile-defense systems and air-traffic control are other fields where MTT is applicable. It can be used in biology, as in [27].

### **1.3 Main Contributions of the Thesis**

The current state of art for distributed multi-target tracking allows MTT to be achieved within acceptable accuracy in real time. But the term acceptable accuracy is subjective and case dependent.

This paper presents a novel model for performing MTT in 3D systems with the assumption of unlimited in-swarm networking. The model introduces different solutions for different problems in MTT while providing an autonomous approach that works individually for each agent without a centralized communication node requirement. The model uses K-means clustering for the belief of targets, called tracks, estimation and JPDA methods for data associations. Also, track estimates are further improved with Kalman filtering.

In addition, we develop a simulation allowing easy testing of model variations for MTT with realistic scenarios. The simulation simulates the members of the swarm, called agents, the observations of each agent, called detections, and the world. The world is simulated as an obstacle-free environment.

The simulation architecture is easily modified and tested with different methodologies. For swarm vs. swarm systems where increasing uncertainty is introduced to the system through nondeterministic motions, increasing numbers of targets, and varying ratios of false positive and false negative detections, this thesis examines the accuracy of overall estimate of the number of tracks and also measures the difference of estimated Cartesian coordinates of each track and the true state of the actual target. The simulation is scalable to large numbers of agents and can be conducted with different parameters.

## **1.4 Organization**

The remainder of the paper consists of four sections. Chapter 2 describes the model formulations including simulation and agent modeling. Chapter 3 identifies key methods used for distributed MTT. Chapter 4 presents the results of the simulation. Chapter 5 concludes paper and provides insight for future works.

## **1.5 Limitations and Assumptions**

### **1.5.1 Assumptions**

This paper presents a model for performing MTT. A simulation is used for testing the performance of the MTT model. The simulation environment is assumed obstacle-free, has predefined reflective borders, and no additional perturbations due to any changes in the environment.

Agents are modeled with a linear state-space perturbed by additive Gaussian noise to represent nondeterministic motions. Also, agents are assumed to have perfect location information, such as via GPS or other localization capabilities. Further, onboard sensing such as with electro-optical cameras are assumed to generate detections. All agents in the system are identical to each other with identical sensors yielding identically structured observation vectors for all agents. Agents have stochastic movements with no collision avoidance where they can cross over each other. Also agents have limited sensor coverage, and the effects of optical target occlusion is approximated in the presented simulation.

### 1.5.2 Limitations

The simulation is not designed for very large systems (e.g., 10,000 vs. 10,000 agents), as for very large systems, there exist some approaches that allow such simulations to be executed efficiently which are not applied in the model implementation presented herein.

In the model, agents acquire tracks from detections (i.e., observations) via K-means clustering. When faced with sparse detections, the model, tested with different approaches for the selection of the best number of clusters or tracks, denoted  $k$ , cannot accurately determine an accurate value for  $k$ . For example, the proposed modified L-method for selecting  $k$  does not work accurately when the number of detections is less than 20. In this manner, the proposed model is best suited for addressing the swarm contexts in question.

Networking costs are typically exponential in the number of agents. As discussed above, treatment of network constraints are left to future works.

Error in GPS accuracy is not represented. Also, accuracy of actual visual systems should be studied. However, imperfect detections are introduced with error rates in order to provide a more realistic approach. Since the real world test results of visual sensors are not available, notional error parameters are used in the simulation. The simulation can easily be tested if the real world test parameters are provided.

THIS PAGE INTENTIONALLY LEFT BLANK

---

## CHAPTER 2:

# Model Formulation

---

Consider a scenario in which two opposing swarm of unmanned aerial systems operating in three dimensions are engaging each other. One of the swarm system is dubbed the ‘friendly swarm,’ with its constituents called ‘agents.’ The entities constituting the other swarm are called ‘targets,’ as illustrated in Figure 2.1.

Before discussing the methods for performing multi-target tracking, we describe the simulation engine constructed to test strategies for performing MTT, which simulates the environment, all agents and targets and their respective trajectories over time, and agent observations (i.e., detections).

The agents’ noisy observations of targets are called detections and are represented as Cartesian coordinates with respect to some globally fixed reference frame. According to the approach presented in this thesis, agents generate their collective belief or representation of targets’ states, known as tracks, from the gathered detections. Each track comprises the agents’ estimates of all target positions and speeds and the estimate error measured by its covariance.

In this thesis, agents maintain their track knowledge individually. Also, agents preserve a history of detections and older tracks, which aids in improving track accuracy. The collective track knowledge of an agent is known as the common operational picture (COP); that is, the COP represents a given agent’s understanding of the current states (positions and speeds) of some or all targets.

## 2.1 Simulation Implementation

The simulation itself is coded with object-oriented design in Matlab. Both simulation and agents in the environment are created as objects. Agent objects comprise both agents, called as Allied Agent, and targets, called as Enemy Agents, with both being inherited classes of the same object. This simulation provides the stochastic trajectories followed by all mobile entities as well as detections of targets. Additionally, implemented algorithms, such as the K-means clustering algorithm and Kalman filtering (described in Chapter 3), are established as objects that are called by the agent itself<sup>1</sup>.

---

<sup>1</sup>The simulation is designed with MATLAB R2012a.

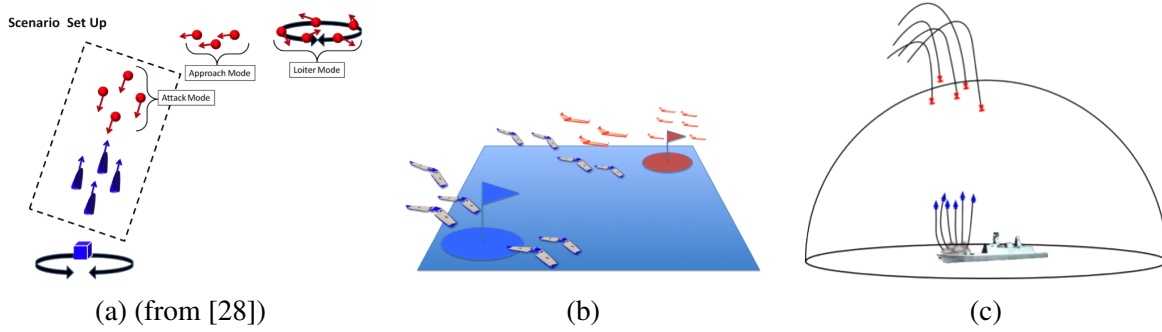


Figure 2.1: Illustrations of the motivating swarm vs. swarm scenario: (a) represents different flight phases for defending UAVs, (b) represents a generic swarm vs. swarm system, and (c) represents a swarm attack and swarm defense scenario for a high valued unit.

The simulation requires three parameters to run: the number of allied agents,  $N_a$ , the number of enemy agents,  $N_e$ , and the total number of turns to run the simulation, that is, the maximum simulation time,  $T_{\max}$ . The simulation feeds each agent with their respective detections and detections acquired by the agent's teammates, shared via the assumed perfect communication network. The general outline of the simulation is portrayed in Algorithms 1 and 2.

---

**Algorithm 1** Simulation (Part-1)

---

```

1: procedure SIMULATIONALGORITHM( $N_a, N_e, T_{\max}$ )
2:   for  $i=1$  to  $N_a$  do
3:      $State \leftarrow \text{RandomState}$  ▷ Agents initialized biased in  $x$  dimension
4:      $AlliedAgents(i) \leftarrow \text{Agent}(state)$ 
5:   end for
6:   for  $i=1$  to  $N_e$  do
7:      $State \leftarrow \text{RandomState}$  ▷ Agents initialized biased in  $x$  dimension
8:      $EnemyAgents(i) \leftarrow \text{Agent}(state)$ 
9:   end for
10:  for  $i=1$  to  $T_{\max}$  do
11:     $\text{CalculateDetections}(AlliedAgents, EnemyAgents)$ 
12:     $\text{UpdateSimulation}(AlliedAgents, EnemyAgents)$ 
13:  end for
14: end procedure

```

---

Algorithm 1 summarizes the workflow of the simulation. Initially, it generates ‘Allied Agents’ and ‘Enemy Agents’ and then, for each time step, the simulation provides the detections to each agent via the `CalculateDetections` function, which is further detailed in Algorithm 2. After each agent acquires and parses its respective set of detections, the simulation updates the state knowledge of the agents in the `UpdateSimulation` function within Algorithm 1.

---

**Algorithm 2** Calculate Detections (Part-2)

---

```
15: procedure CALCULATEDETECTIONS(AlliedAgents, EnemyAgents)
16:   for i=1 to sizeof (AlliedAgents) do                                ▷ For each AlliedAgent do
17:     NewDetections  $\leftarrow$  ParseDetections (AlliedAgents(i), AlliedAgents, EnemyAgents)
18:     AlliedAgents(i).GatherDetections(NewDetections);
19:   end for
20:   for i=1 to sizeof (AlliedAgents) do
21:     for j=1 to sizeof (AlliedAgents except AlliedAgents (i)) do
22:       NetworkDetections  $\leftarrow$  AlliedAgents (j).Detections
23:     end for
24:     AlliedAgents(i).GatherNetworkDetections(NetworkDetections);
25:   end for
26: end procedure
```

---

‘Allied Agents’ acquire their detections via Algorithm 2. This algorithm uses the ParseDetections function, detailed in Algorithm 3, to generate the detections of the agent that can be perceived by agents’ sensors. These detections are those in the field of view (FOV) of the agent but not occluded by any other entity in the environment. The detections that the agent acquires via networking, referred to as ‘network detections,’ are provided in the next step. These network detections comprise all of the detections of all agents (since a perfect and complete communication network is assumed).

It should be kept in mind that each agent itself first processes its own individual detections, then acquires its network detections within the next iteration of the loop. In real world application, it is assumed that agents will acquire their locally generated detections first and network detections second in each time step. This process is depicted in Algorithm 2. The detection model used for generating detections can be seen in Equation (2.1) where  $\mathcal{N}(\mu, \lambda)$  is Gaussian noise with mean  $\mu$  and standard deviation  $\lambda$  added to the exact state information of the true target state.

$$Detection(t) = Target\_State(t) + \zeta, \quad \zeta \sim \mathcal{N}(\mu, \lambda) \quad (2.1)$$

The simulation uses Cartesian coordinates for state representations but FOV and occlusion calculations are done according to spherical coordinates relative to the given agent. The orientation of spherical coordinates in terms of the inertial Cartesian coordinate reference frame is illustrated in Figure 2.2. For spherical coordinates, azimuth, elevation and  $r$  are computed as  $\theta$ ,

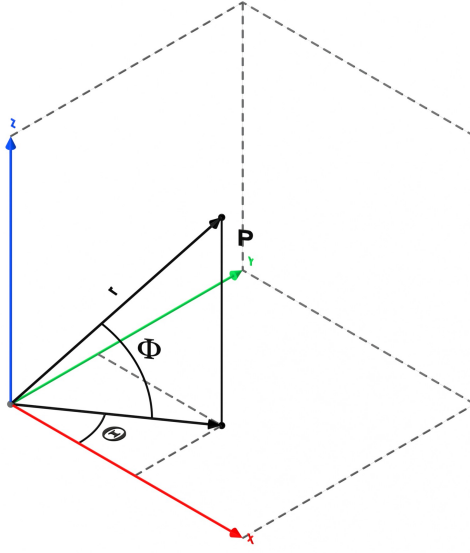


Figure 2.2: Spherical coordinates description with Cartesian coordinates.

$\phi$ , and distance, respectively. In FOV and occlusion computations for an agent, the agent itself lies at the origin of the local frame, and any other object's spherical coordinates are calculated relative to the current agent.

It should be kept in mind that detections are generated according to the exact locations of the targets. If the exact location of the target is not occluded, the detection is generated even if the detection's location is occluded. The simulation only currently checks the target location for occlusion, rather than validating the feasibility of the detection. Higher fidelity simulations may address this approximation in future studies.

The detection generation is handled by Algorithm 3. Detections, according to (2.1), are enemy agents that fall in a FOV of a selected agent and not occluded by any other agent in the simulation. Algorithm 3 works for each agent individually. It requires the following inputs: target agent as 'Agent', all allied agents as 'AlliedAgents' and all enemy agents as 'EnemyAgents'. The output of Algorithm 3 is detections that satisfy all the required conditions; not occluded, FOV of the agent and with Gaussian error.

For this process, first, possible detections are computed. These possible detections are the enemy agents that reside in FOV of the target agent. For each enemy agent, their spherical coordinates relative to target agent are computed and if these parameters are less than or equal

to predetermined parameters, these enemy agents are accepted as possible detections. This process is handled by the ‘FindDetections’ function within Algorithm 3.

---

**Algorithm 3** Parse Detections (Part-3)

---

```

27: procedure PARSEDETECTIONS(Agent, AlliedAgents, EnemyAgents)
28:   PosDetections  $\leftarrow$  FindDetections (Agent, EnemyAgents)  $\triangleright$  Possible Detections
29:   if PosDetections  $\neq \emptyset$  then
30:     BlockingAgents  $\leftarrow$  (EnemyAgents  $\cup$  AlliedAgents except Agent)  $\in$  FOV(Agent)
31:     OccDetections  $\leftarrow$  OcclusionDetections (Agent, BlockingAgents, PosDetections)
32:     if OccDetections  $\neq \emptyset$  then
33:       Return Detections  $\leftarrow$  AddError (OccDetections, Agent)  $\triangleright$  Adds Gaussian noise
34:     else
35:       Return Detections  $\leftarrow \emptyset$ 
36:     end if
37:   else
38:     Return Detections  $\leftarrow \emptyset$ 
39:   end if
40: end procedure

```

---

After possible detections are generated, possible agents that may occlude any detection are stated to represent blocking agents. The vector of this list of blocking agents contain the index of all allied agents and targets that reside in the FOV of the agent in question, and is an input to the function *OcclusionDetections*, detailed in Algorithm 4, and returns all detections that are not blocked geometrically by any agent.

After all detections are computed via Algorithm 4 with occlusion taken into account, if there still exist detections, these detections are further modified according to (2.1) via the *AddError* function of Algorithm 4. At the end, Algorithm 4’s output is the detections which only has the mean of each detection. It is quite clear that these detections become input parameters of each agent via Algorithm 2.

After all detections are computed via Algorithm 4 with occlusion taken into account, if there still exist detections, these detections are further perturbed according to Equation (2.1) via the *AddError* function of Algorithm 4. At the end, Algorithm 4’s output is the vector of detections which represents the true locations of each detection. It is clear that these detections become the input parameters for each agent in Algorithm 2.

The occlusion computation, represented in Algorithm 4, requires the following inputs, namely the agent whose perspective is considered, a vector of all agents and targets that lie within

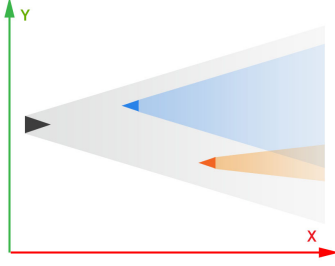


Figure 2.3: An illustration of geometric occlusion in generating detections. The red and blue agents (right) block portions of the field of view of the black (left) agent.

the given agent's FOV, and a vector of possible detections called *PosDetections*. Algorithm 4 iterates over all agents and targets within the given agent's FOV. The spherical coordinate differences between the given agent and the  $i^{\text{th}}$  entity in this list is computed and denoted  $\theta_{Blocker}$ ,  $\phi_{Blocker}$  and  $r_{Blocker}$ . The occlusion parameters representing the angular windows which define occlusions are  $\theta_{Occ}$  and  $\phi_{Occ}$ . This process allows the simulation to have generic occlusion parameters according to the relative location of the occluding entity. For each entity, each possible detection is verified whether or not it is occluded, that is, possible detections' spherical coordinates relative to the given agent are computed as  $\theta_{Det}$ ,  $\phi_{Det}$  and  $r_{Det}$ , from which the occlusion thresholds are evaluated. If a possible detection is occluded, it is removed from the possible detections list, which prevents unnecessary computations over irrelevant detections. The output of Algorithm 4 is a distilled vector of detections that are free from occlusion by any entity in the environment.

The *AddError* function adds Gaussian noise with mean  $\mu$  and covariance  $\lambda$  to each detection according to Equation (2.1). In order to generate appropriately rotated noise parameters (since measurement error is relative to the agent's position and orientation), Equation (2.2) is used. The simulation has predetermined noise parameters for predetermined target Cartesian locations relative to the agent itself. The main idea in *AddError* is to generate accurate error parameters according to the given target location via finding  $\theta$ ,  $\phi$  and  $r$  parameters and applying the method described in Section 2.1.1.

First, the relative spherical coordinate differences,  $\theta$ ,  $\phi$  and  $r$  parameters of the target relative to the agent, are computed according to their Cartesian coordinates. Since the simulation loads in predefined noise parameters for a given target location, the difference between these predefined

---

**Algorithm 4** Occlusion Detections (Part-4)

---

```
41: procedure OCCLUSIONDETECTIONS(Agent, BlockingAgents, PosDetections)
42:   for i=1 to sizeof (BlockingAgents) do                                ▷ For each BlockingAgents do
43:     [ $\theta_{Blocker}$ ,  $\phi_{Blocker}$ ,  $R_{Blocker}$ ]  $\leftarrow$  SphericalDifference (Agent, BlockingAgents(i))
44:     [ $\theta_{Occ}$ ,  $\phi_{Occ}$ ]  $\leftarrow$  OcclusionParameters (Agent, BlockingAgents(i), [ $\theta_{Def}$ ,  $\phi_{Def}$ ])
45:     for j=1 to sizeof (PosDetections) do                                ▷ For each PosDetections, check
46:       [ $\theta_{Det}$ ,  $\phi_{Det}$ ,  $R_{Det}$ ]  $\leftarrow$  SphericalDifference (Agent, PosDetection(j))
47:       if ( $R_{Blocker} < R_{Det}$ ) & ( $|\theta_{Blocker} - \theta_{Det}| \leq \theta_{Occ}$ ) & ( $|\phi_{Blocker} - \phi_{Det}| \leq \phi_{Occ}$ ) then
48:         PosDetections  $\leftarrow$  PosDetections - PosDetections(j)
49:       end if
50:     end for
51:   end for
52:   Return OccDetections  $\leftarrow$  PosDetections
53: end procedure
```

---

parameters and newly calculated  $\theta$ ,  $\phi$  and  $r$  parameters of the target can be computed and denoted  $\theta_{diff}$ ,  $\phi_{diff}$  and  $r_{diff}$ . These three parameters becomes the input for Section 2.1.1 with a 3D noise vector (representing noise in position values). The output of Equation (2.2) is a rotated noise vector which represents the input noise parameters of the AddError function for each dimension according to Equation (2.1).

### 2.1.1 Spherical Rotation of Vectors in 3D

The simulation requires the ability to compute the spherical rotation of a vector in various parts of the simulation. Spherical rotation of a vector is calculated according to Equation (2.2) where a given vector  $v \in \mathbb{R}^3$ , and rotation angles  $\theta$  and  $\phi$  are inputs and the rotated vector  $v_{rotated} \in \mathbb{R}^3$  is the output. The rotation matrices governing the rotations in  $\theta$  and  $\phi$ , respectively, are denoted  $M_1$  and  $M_2$ , and represent counter clockwise rotations. The  $(\cdot)'$  notation represents the transpose operator.

$$M_1 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (2.2)$$

$$v_{rotated} = M_1 \cdot M_2 \cdot v \cdot M_2' \cdot M_1'$$

For any given  $\theta$  and  $\phi$  angles, Equation (2.2) rotates any column vector  $v \in \mathbb{R}^3$ . This process

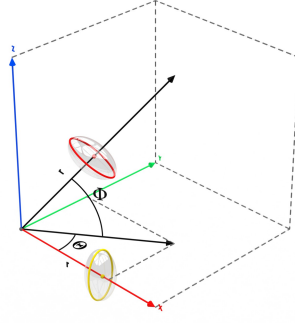


Figure 2.4: An example of spherical rotation. The yellow circle (positioned lower middle) represents the original vector and red circle (positioned upper middle) represents the rotated vector with angles  $\theta$  and  $\phi$ .

is used by the simulation in a number of calculations where the input vector consists of the parameters required by the simulation.

## 2.2 Agent Modeling

### 2.2.1 Motion Modeling

Agents can be modeled via a linear state-space model for constant speed perturbed with Gaussian noise. Each agent is stored according to the Cartesian coordinate system with 3D world coordinates and a 3D speed vector. Also it should be kept in mind that agents bounce back from the predetermined simulation borders as described in Section 2.3. The observation model of each agent's state is described in Equation (2.3). This observation model is used by simulation in order to update state of the agents, which is also represented in [29].

$$x(t+1) = A \cdot x(t) + v, \quad v \sim \mathcal{N}(\omega, \rho) \quad (2.3)$$

In Equation (2.3),  $\mathcal{N}(\omega, \rho)$  is Gaussian noise of the motion with mean  $\omega$  and covariance  $\rho$ .

The state dynamics matrix, called  $A$ , and the state of an agent at time  $t$ , called  $x(t)$ , are defined in Equation (2.4) and Equation (2.5), respectively.

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

$$x(t) = (Loc_x(t), Loc_y(t), Loc_z(t), V_x, V_y, V_z)' \quad (2.5)$$

Note that for the discrete time models, the time interval between  $t$  and  $t - 1$  is assumed to be one. For example, then, using the above definitions,  $Loc_x(t + 1) = Loc_x(t) + 1 \cdot V_x$ .

The simulation updates agents state via the `UpdateSimulation` function of Algorithm 1 according to Equation (2.3). It should be kept in mind that spherical rotation described in Section 2.1.1 plays an important role in detection generation but not in motion modeling of the agent itself (since agent motion is in the global reference frame whereas detections are relative to the local frame).

### 2.2.2 Sensor Modeling

For MTT, each measurement originates from at most one target. Some sensors may not provide measurements at every time interval. Some measurements may arise from targets, some from clutter and some targets may not yield any measurement at all in any particular time interval.

Measurement error characteristics are assumed to be identical for each sensor. Each detection is assumed to be acquired from sensors, such as electro-optical cameras mounted on UAVs. The detections potentially include observation of enemy agents that fall within the FOV of the agent and are not occluded by any other entity in the environment, generated with Gaussian noise as in Equation (2.1).

The detections of the agents are provided by the simulation, which allows modular study of the MTT approaches separate from specific sensor models, and each agent parses these detections according to their own process detailed in Section 3.1. Simulation provides both agents' own detections and the detections agents acquire via networked communication with teammates, as per Algorithm 2.

Each agent stores each detection with the three elements shown in Equation (2.6); the 3D location of each detection is called  $\bar{d} \in \mathbb{R}^3$ , the covariance or uncertainty matrix is called  $\Sigma_d \in \mathbb{R}^{[3,3]}$ , and the index of the detection (stored for post simulation analysis). For detections assumed acquired local to the given agent, the simulation provides  $\bar{d}$  of each detection, where  $\bar{d}$  is the detection's Cartesian coordinates.  $\Sigma_d$  of a detection is computed by the agents itself, which accounts for noise in each Cartesian dimension relative to the  $\bar{d}$ . The stored detection index represents the relationship to assigned tracks (as described later in Section 3.4.1 and Section 3.4.2). The  $\Sigma_d$  of a local detection is computed via Equation (2.2), with the following inputs: the error vector defining default noise characteristics for each dimension, and  $\theta$  and  $\phi$  representing the spherical coordinate rotations relative to the agent's detection.

$$\begin{aligned} \bar{d} &= (Loc_x, Loc_y, Loc_z), \\ \Sigma_d &= \begin{bmatrix} Err_{xx} & Err_{xy} & Err_{xz} \\ Err_{yx} & Err_{yy} & Err_{yz} \\ Err_{zx} & Err_{zy} & Err_{zz} \end{bmatrix} \end{aligned} \quad (2.6)$$

It merits noting that the simulation generates a detection, which is provided to the agent, according to its own model of the uncertainty parameters (though still based on the true target's location), whereas the agent processes this detection and constructs its covariance matrix based on its own model of the noise parameters. Though for the presented work, the agent is assumed to have an accurate model of the uncertainty, this flexibility allows for future study to examine the impact of modeling errors in sensor characteristics.

Each agent acquires detections of its allies as network detections. These network detections have their respective  $\bar{d}$  and  $\Sigma_d$ , where  $\Sigma_d$  is the covariance matrix for each detection as computed by the originated teammate and passed along to the receiving agent. As a result, agents do not compute  $\Sigma_d$  for network detections.

## 2.3 Environment Modeling

The simulation environment is obstacle free and obstacle avoidance is not considered in this work. Agents are assumed to be able to cross over each other but are constrained to maneuver wholly within the simulation borders, from which they simply bounce back.

Recall that the state of an agent is defined by Equation (2.5). The initial locations for each agent and target within the simulation are determined according to a normal distribution in space,

centered about  $(Loc_x, Loc_y, Loc_z)'$  defined for each entity. This bias parameter can allow for investigating different initial configurations of agents and targets, though a thorough exploration of their impact is reserved for future study.

The agents and targets tend to move towards each other according to Equation (2.3). This is achieved via biases in  $V_x$ ,  $V_y$  and  $V_z$ . These three parameters represent the speed vector of an agent, and by appropriate specification (e.g.,  $V_x$  for agents,  $-V_x$  for targets), agents and targets can be simulated to be in approaching trajectories. The above initialization biases are predefined in the simulation. The main highlight is the flexibility of the simulation to enable deployment of agents and targets clustered together and/or in opposite locations.

According to Equation (2.3), at each time step, the simulation algorithm calculates the agent state for time  $t + 1$  and if agent exceeds the predetermined simulation borders, the speed element(s), e.g.,  $V_x$ ,  $V_y$ ,  $V_z$  of Equation (2.5), are negated in the dimensions where the agent violated the simulation border. This process prevents agents from exceeding predefined simulation borders to ensure that over time, a fixed and known number of potential targets remain within the area of operations.

THIS PAGE INTENTIONALLY LEFT BLANK

---

## CHAPTER 3:

# Distributed Target Tracking from Multiple Sensors

---

The important issue in performing MTT is generating an accurate common operational picture (COP) for each agent. The strategy for generating observations (detections) and COPs for each agent is described in this section.

### 3.1 General Flow

According to the simulation, there exists two types of agents; one called ‘Allied Agents’ and the other one called ‘Enemy Agents.’ Both of these agents are the same object, but ‘Enemy Agents’ have only their state updated at each time step. In this chapter, we present the information flow of the ‘Allied Agent’ (which, in the object-oriented sense, automatically includes ‘Enemy Agents’). For succinctness, ‘Allied agents’ are referred as ‘agents’ in the remainder of the discussion presented below.

In this thesis, we assume that each agent acquires its own detections via an onboard sensor, such as a camera, and also obtains the detections of its allies or teammates via the interconnecting network. Since a perfect network is assumed (that is, no loss, delay, or error in transmissions), all agents are assumed to know all other agents’ detections and able to immediately parse and process them using locally executed algorithms. In this manner, this chapter describes the distributed target tracking methods for an individual agent in detail, and provides measures of performance both at the individual agent and team levels.

Figure 3.1 graphically illustrates the information flow for an agent for each time step. Each agent acquires first only its State and Detections from the simulation, as it would in physical settings from its *proprioceptive* sensors (e.g., GPS, inertial sensors) and *exteroceptive* sensors (e.g., electro-optical camera), respectively. Other than these inputs and those representing shared detections from networked teammates, called network detections, all subsequent calculations are assumed internal to each agent.

The agent first priority is to generate the Detections from the knowledge it acquired from the simulation which is explained in Section 3.2. After all Detections are acquired, the tracks are generated as explained in Section 3.3. In this state of the information flow, the agent has generated its tracks and has the knowledge about which tracks originated from which Detections.

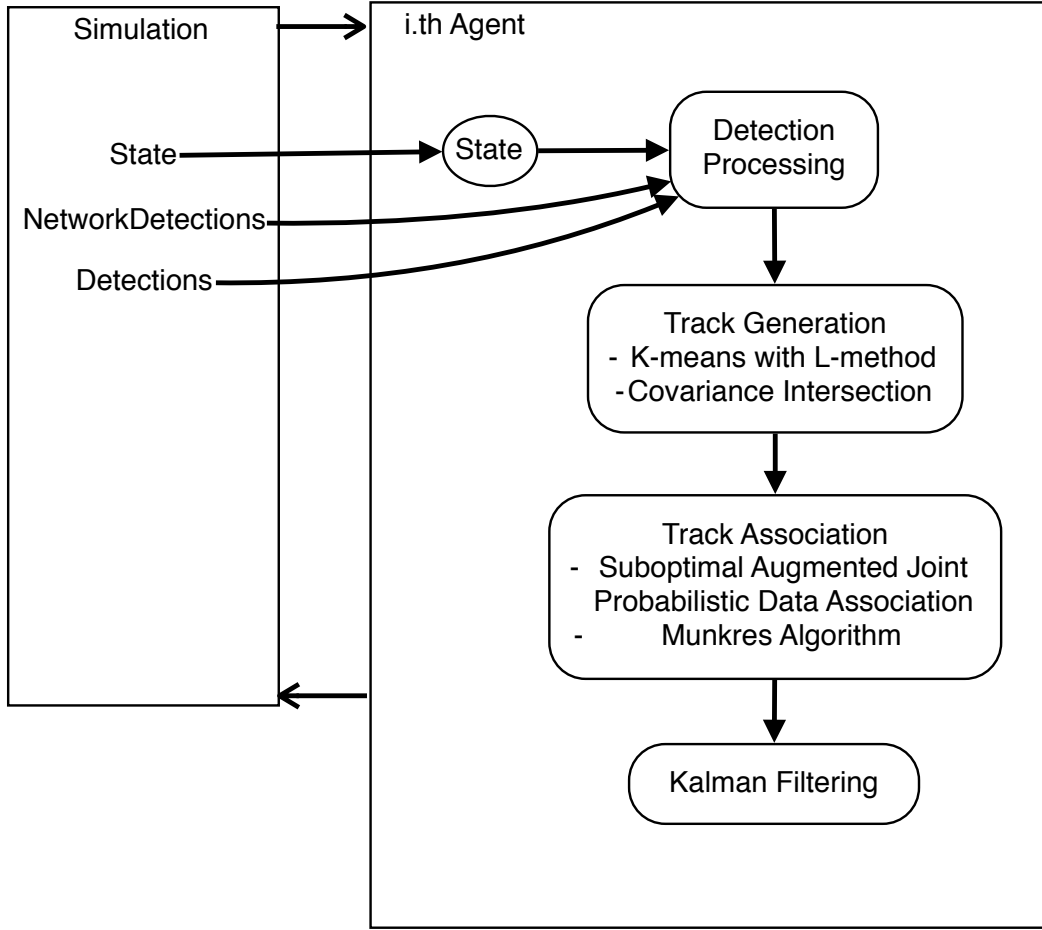


Figure 3.1: General information flow of an agent. State, Detections, and NetworkDetections are generated by the simulation and provided as input to the agent. This figure depicts the model tested for any discrete time step.

Then, the tracks at time  $t$  are associated with the tracks at time  $t - 1$  via ASJPDA, as explained in Section 3.4. This process allow us to carry out the KF parameters which is detailed in Section 3.5. KF provides better target estimations and an important process for improving the accuracy of the tracks.

There exists different procedures for MTT. The algorithms that uses these approaches are detailed in later sections. In here, the Algorithm 5 describes the three methods used for two purposes. These purposes do not tie to any specific methods, so they are described in this section.

In the Algorithm 5, the method ‘AgentAlgorithm’ is referred by the simulation in order to provide state of the agent, which allow agent itself to acquire its state knowledge. The ‘AgentUpdate’ in in Algorithm 5 is called by the agent itself at the end of each time step. This method allow agent to store its current parameters in ‘History’ variable, up to a certain limit, via method ‘UpdateHistory’.

---

**Algorithm 5** Agent

---

```

1: procedure AGENTALGORITHM(State)
2:    $State \leftarrow State$ 
3:    $Detections \leftarrow \emptyset$ 
4:    $Tracks \leftarrow \emptyset$ 
5: end procedure
6: procedure AGENTUPDATE(NewState)
7:    $UpdateHistory()$ 
8:    $State \leftarrow NewState$ 
9:    $Detections \leftarrow \emptyset$ 
10:   $Tracks \leftarrow \emptyset$ 
11: end procedure
12: procedure UPDATEHISTORY
13:   $NewBatch \leftarrow State \cup Detections \cup Tracks$ 
14:   $History \leftarrow History \cup NewBatch$ 
15:  if  $size(History) > MaxSizeAllowed$  then
16:     $History_{Oldest} \leftarrow \emptyset$ 
17:  end if
18: end procedure

```

---

The methods used by the agent in order to process detection, generate tracks and improve track estimates are further detailed in the remaining sections. The order of these processes are visualized in Figure 3.1.

### 3.1.1 Notation and Definitions

Given the need for integration of multiple algorithmic approaches, this section identifies and defines the notation to be used throughout the formulation. These notations can be seen in Table 3.1, Table 3.2 and Table 3.3

Variable	Description	Dimension
$N_a$	Number of ‘Allied Agents’	scalar
$N_e$	Number of ‘Enemy Agents’	scalar
$T_{max}$	Number of turns the simulation runs	scalar
$\zeta(\mu, \lambda)$	The Gaussian distribution for detection generation	$\mu$ and $\lambda$ are scalar
$v(\omega, \rho)$	The Gaussian distribution for <i>agent</i> state update	$\omega$ and $\rho$ are scalar
$N_n(\chi, \sigma)$	The probability distribution (from Section 3.4.1.1)	$\chi$ is $1 \times n$ and $\sigma$ is $n \times n$
$N_n(\chi, \sigma)$	The probability distribution (from Section 3.4.1.2)	$\chi$ is $1 \times n$ and $\sigma$ is $n \times n$
$y$	Current measurement (from Section 3.4.1.2)	$1 \times n$
$\theta$	Spherical coordinate horizontal angle	radian
$\phi$	Spherical coordinate vertical angle	radian
$r$	Spherical coordinate distance	scalar
$\bar{d}$	the cartesian coordinates of the detection	$1 \times 3'$
$\Sigma_d$	the belief of error for the detection	$3 \times 3$
$\bar{t}$	the cartesian coordinates of the target	$1 \times 6'$
$\Sigma_t$	the belief of error for the target	$3 \times 3$

Table 3.1: Table of notations and definitions, including relevant vector or matrix dimensions

## 3.2 Detection Processing

### 3.2.1 Agent Detections

The simulation provides the cartesian coordinates of each detection to the related agent. The agent itself generates a variable called Detections stores the cartesian coordinate of each detection as the mean, called  $\bar{d}$ .

The agent requires 2 different parameters per detection. The mean,  $\bar{d}$ , of the detection is provided by the simulation. The second parameter, the covariance of the detection representing the uncertainty in the detection measurement, called  $\Sigma_d$ , is generated by the agent itself. The  $\bar{d}$  and  $\Sigma_d$  are described in Equation (2.6).

The agent constructs the  $\Sigma_d$  of a detection as follows; there exists a predetermined covariance, called  $\Sigma_{\bar{d}}$ , for a predetermined cartesian location relative to the agent. First, the relative cartesian location of the detection is computed. Since the detection’s relative location and the  $\Sigma_{\bar{d}}$ ’s relative location is known, the spherical coordinates of these two locations computed. The difference of these two locations provides the angular parameters that is needed to compute  $\Sigma_d$ . According to Section 2.1.1,  $\Sigma_{\bar{d}}$  is rotated with the computed angles and the resulting matrix from Section 2.1.1 becomes  $\Sigma_d$  of the detections.

Variable	Description	Dimension
$N$	number of state estimates (from Section 3.3.2)	scalar
$\hat{x}_i$	mean of state estimate (from Section 3.3.2)	$1 \times 3$
$P_i$	covariance of state estimate (from Section 3.3.2)	$3 \times 3$
$w_i$	weighting coefficients (from Section 3.3.2)	scalar
$\hat{x}_0$	fused mean of state estimate (from Section 3.3.2)	$1 \times 3'$
$P_0$	fused covariance of state estimate (from Section 3.3.2)	$3 \times 3$
$C_i$	# of centroids (from Section 3.3.1)	scalar
$\mu_i$	the data of the centroid (from Section 3.3.1)	$1 \times 3$
$x_j$	the data of the observation (from Section 3.3.1)	$1 \times 3$
$n$	length of the observations (from Section 3.3.1)	scalar
$n^*$	length of the true positive detections (from Section 3.3.1)	scalar
$n^+$	length of the targets (from Section 3.3.1)	scalar
$P$	the coefficients of the function (from Section 3.3.1.1)	$1 \times 3$
$P$	the coefficients of the function (from Section 3.3.1.2)	$1 \times 2$
$RMSE_n$	The margin of error (from Section 3.3.1.3)	scalar
data	Original data (from Section 3.3.1.4)	$x \times y$
$data^*$	Modified data (from Section 3.3.1.4)	$x^* \times y^*$
$z^*$	length of tracks at time $t - 1$ (from Section 3.4.1)	scalar
$z$	length of tracks at time $t$ (from Section 3.4.1)	scalar
$\beta_{t,j}$	the likelihood of $track_t$ with $hit_j$ (from Section 3.4.1)	$n \times m$
$\beta_{0_t}$	the likelihood of $track_t$ not associated with any $hit_j$ (from Section 3.4.1)	$n \times 1$
$Results_M$	Association results (from Section 3.4.2)	$n + m \times 1$
$Results_M^*$	Modified association results (from Section 3.4.2)	$n + m \times 1$

Table 3.2: Table of notations and definitions, including relevant vector or matrix dimensions

The Algorithm 6 handles to process described above. In Algorithm 6, the Detections acquired by the agent are assigned to the  $\vec{d}$ . The  $\Sigma_d$  of each detection is calculated according to Section 2.1.1, resulting the Equation (2.6), as in the Section 2.2.2. This process is handled by ‘ComputeCov’ method. There exists no network Detections yet.

In current state of the agent, all Detections are accepted as tracks without any computation. This is due to sensor modeling of Detections where each detection can only originate from at most one target, so each detection is already a track and if it is a false positive Detections, there is no way distinguish it with current state of the information.

Variable	Description	Dimension
$\tilde{x}(k+1)$	prior estimates of the mean (from Section 3.5)	$1 \times 3'$
$\tilde{\Sigma}(k+1)$	prior estimates of the covariance (from Section 3.5)	$3 \times 3$
$K$	Kalman gain	scalar
$\hat{x}(k+1)$	posterior estimates of the mean (from Section 3.5)	$1 \times 3'$
$\hat{\Sigma}(k+1)$	posterior estimates of the covariance (from Section 3.5)	$3 \times 3$
$R$	Estimated measurement errors	$6 \times 6$
$Q$	Dynamic noise matrix	$6 \times 6$

Table 3.3: Table of notations and definitions, including relevant vector or matrix dimensions

---

#### Algorithm 6 Agent

---

```

19: procedure GATHERDETECTIONS(NewDetections)
20:    $\bar{d} \leftarrow \text{Newdetections}$ 
21:    $\Sigma_d \leftarrow \text{ComputeCov}(\text{Detections})$ 
22:    $\text{Tracks} \leftarrow \text{Detections}$ 
23: end procedure

```

---

### 3.2.2 False Detections

The simulation has two predetermined parameters for generating false positive and false negative detections. These parameters are used as a threshold value in order to generate false detections.

The false negative detections, which means simulation says there exists no detection while it should be, is generated per detection of each agents. If an agent has  $n$  detections for time  $t$ , these detections are verified prior to feeding the agent itself. For each detection, a uniformly random number is generated. If the uniform random number is less than or equal to the false negative threshold value, that detection is not provided to the agent itself. This process is executed for each detection, for a total of  $n$  times.

The false positive detections, that is, the simulation returns a detection where one should not be present, are generated per each detection of each agent. For each detection, a uniformly random number is generated and if this number is less than or equal to the false positive threshold value, a new feasible (but otherwise false) detection is appended as another detection to the Detections list of the agent. This process is executed per true detection, which may result in at most double the number of true positive detections in worst case.

These false detections are generated at each time step, for each agent, for each detection the agent has for the given time  $t$ .

### 3.2.3 Distributed Sensing with Network Detections

Next, the agent receives (from the simulated network) the set of all detections and their respective covariances from all allied agents.

The Algorithm-7 handles network detections. Network detections of the agent are all the Detections that its allies acquired at time  $t$ . There is a possibility that an agent can acquire no network detections if its allies have no detections themselves. It should be noted that these network detections are appended to the current Detections lists and all of these are stored as Detections by the agent itself.

---

#### Algorithm 7 Agent

---

```

24: procedure GATHERNETWORKDETECTIONS(NetworkDetections)
25:    $[\bar{d}, \Sigma_d] \leftarrow \text{NetworkDetections}$ 
26: end procedure

```

---

From this point on, the simulation does not provide any more information to the agent until the next time step.

## 3.3 Track Generation

After all Detections are acquired in previous steps, agent calls K-means method, explained in Section 3.3.1. In general, K-means acquires the Detections and generates tracks, i.e., the Cartesian location of the tracks, each denoted  $\bar{t} \in \mathbb{R}^6$ , from these Detections. After tracks are generated, the covariance intersection method, explained in Section 3.3.2, is called in order to generate the belief matrix of error, called  $\Sigma_t \in \mathbb{R}^{[3,3]}$ , of each track. The belief error matrix is generated as the intersection of the  $\Sigma_d$  of the detections that the track is originated from. The covariance of the tracks is denoted  $\Sigma_t \in \mathbb{R}^{[3,3]}$  of the tracks while the Cartesian location of the tracks, each called  $\bar{t} \in \mathbb{R}^6$ .

The Algorithm-8 handles to process described above. The purpose of Algorithm-8 is to generate tracks from Detections via K-means method, described as Section 3.3.1. At the end of the ‘KMeansClustering’ function, each track acquires its  $\bar{t}$ . The covariance of the tracks are generated according to the ‘CovarianceAssessment’ function, works according to Section 3.3.2. As a result, at the end of Algorithm 7, all tracks of the agent is generated with their  $\bar{t}$  and  $\Sigma_t$ .

---

**Algorithm 8** Agent

---

```
27: procedure TRACKGENERATION(Detections)
28:   [detections.Index,  $\tilde{t}$ ]  $\leftarrow$  KMeansClustering (Detections)
29:    $\Sigma_t \leftarrow$  CovariancesAssessment (Detections)  $\triangleright$  Determines covariance of tracks
30: end procedure
```

---

The methodologies use in track generation is explained below. It should be noted that there exists different approaches to handle the similar problems, but the methodologies described below are used in this paper.

### 3.3.1 K-Means Clustering

K-means clustering is a method for grouping given data points in the environment according to some distance criterion. These data points may represent observations, test results or some other information based on the application. The dimensionality of the data points may vary according to application which means each data point is an element in  $\mathbb{R}^m$ , where  $m$  is arbitrary but same for all data. The only issue with increased dimensionality is the computational requirements. K-means clustering requires the objective number of partitions, called centroids, in order to cluster the data points. It can be deduced that, for the objective of  $k$  centroids  $C_1 \dots C_k$  with given  $n$  data points as  $x_1 \dots x_n$ , K-means clustering finds the optimum number of centroids according to Equation (3.1). In Equation (3.1), the square of Euclidian distance is used for distance computation which is the case, but not restricted, in most of the applications.

$$\min \left( \sum_{i=1}^k \sum_{x_j \in C_i} \|x_j - \mu_i\|^2 \right) \quad (3.1)$$

K-means clustering basically picks  $k$  random centroids initially and updates their location with the nearest observation in each iteration while each centroid center is the mean of all observations belongs the same centroid. This process iterates until no observation is left unassigned to a centroid. K-means clustering methods may vary according to the application, but K-means clustering description provided above with Euclidean distance is used in this paper.

In this paper, K-means clustering is used for track generation. Prior to K-means clustering all *Detections* of the agent, both from its own sensors and relayed from its allies, are perceived, but not processed. It should be kept in mind that *Detections* acquired by its own sensors have  $\Sigma_d$  in terms of agent itself while *Detections* acquired from its allies have their own  $\Sigma_d$  according to

their originated agent. As a result, all Detections in this state of the agent workflow have their  $\bar{d}$  and  $\Sigma_d$  and ready to generate tracks.

In order to generate accurate tracks from Detections, the optimum way for selecting best  $k$  needs to be analyzed. The performance of K-means mainly depends on selecting optimum  $k$ . The method for selecting best  $k$  affects the overall performance of the simulation. Since the exact number of tracks in the environment is unknown, without accurate best- $k$  selection method, there is no way for finding accurate number of tracks which hampers the overall performance.

Selecting best  $k$  for K-means clustering is a hard problem and investigated by [11–13]. In general, all methods for selecting the best  $k$  rely on exhaustive tries of K-means approach with  $k$  varying from two to the maximum number of observations and analyzing these results to select best  $k$ . This thesis investigates several similar approaches for this best  $k$  selection. According to the simulation, data for K-means are detections in 3D, i.e., in  $\mathbb{R}^3$  and each detection has same characteristics for noise estimation; all agents have the same characteristic sensor and each detection has the same characteristics with similar noise error estimates that vary according to relative observation point.

In order to decide on best  $k$ , three different approaches are tested against each other with the same input. The input is generated for K-means method, where  $k$  varies from two to total number of observations and the input is, for each  $k$ , total number of centroids versus the sum of total distance to associated centroid center for each observation. After these parameters are generated, all three methods provide their own  $k$  estimation.

Table 3.4 is an example situation for four observations. In this case, there exists total of 4 observations and  $k$  varies 2...4. The table shows the results for  $k = 2$  with the distance of each observation to its associated centroid. In order to test the best- $k$  selection algorithms, total distance parameter of given  $k$  is send as input for all  $k$  values. According to Table 3.4, 6.29 is send with  $k = 2$  as a part of input. The input consists of all distances for each  $k$ .

For any number of data, or observations, provided as input, K-means clustering method is called for  $k$  varies from two to maximum number of observations. When the K-means clustering method is called with this methodology, the overall graph for any number of observation is similar, but not the same, as long as the graph is depicted with  $k$  and  $\Sigma(Distance)$  where distance is each observation Euclidean distance to its own centroid. When the graph is analyzed, for independent number of clusters or data points, there exists always a curve with minor oscillations.

Observations			Centroid			Distance
x	x	z	x	y	z	
1	1	1	3	3	3	3.46
3	3	3	3	3	3	0
5	5	5	3	3	3	3.46
15	15	15	15	15	15	0

Table 3.4: For observations  $\in \mathbb{R}^3$  with total of 4 observation,  $k$  is selected starting from  $2 \dots 4$ . In this table,  $k = 2$ . Each observation is associated with one of the centroids. Total distance in  $k = 2$  is  $\sum Distance = 6.92$

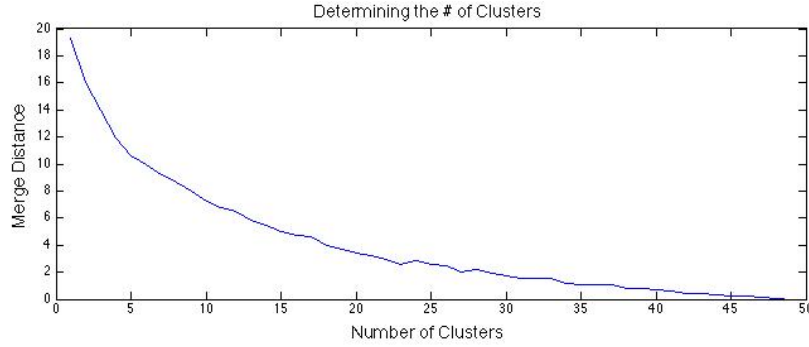


Figure 3.2: A sample graph for number of clusters against total distance to associated centroids for all observation.

These oscillations do not affect the overall slope of the curve, but the slope in minor sections are affected. This curve can be seen in Figure 3.2. All methods described below try to use similar graphs to find the best  $k$ . Since the curve does not have the same slope, it is not reasonable to use the second derivative of the observations to find the best  $k$ .

The observation points are generated according to Equation (2.3) where the same approach is used by simulation for generating detections from tracks. This approach allows us to create more realistic observations. The data for analyzing best  $k$  methods, for total of  $n$  observations (Detections), are generated as follows;

- Step 1: 10% of all Detections, which is equal to  $0.1n$ , are generated as false positive Detections. The number is rounded if needed. False positive Detections are randomly generated Detections that are not originated from *targets*.
- Step 2: The number of false positive detections are subtracted from  $n$ , resulting in the number denoted as  $n^*$ . These Detections need to originate from *targets*. As a result, it is accepted that the ratio between target and Detections is 1 to 10. Due to this assumption,

10% of  $n^*$  is accepted as the number of *targets*, and the resulting number, called  $n^+$  is rounded if needed.

- Step 3: For each target  $t$ , one detection is generated according to Equation (2.3).<sup>2</sup>This process is repeated for each target until the number of detections equal to  $n^+$ .
- Step 4: Merge real Detections with false positive Detections. It should be noted that  $0.1n + n^* = n$ .
- Step 5: Call K-means clustering algorithm for  $k = 2 \dots n$ . Generate *data* matrix which is  $n \times 2$  where first column consists of current value of  $k$  and second column consists of  $\sum_{i=1}^n (distance_i)$  where distance is Euclidian distance of Detections to their associated centroid.

According to this methodology, each detection, if it is not a false positive detection, is originated from a true target and detections originated from same target clustered as a result. This situation is similar to the real world scenario where Detections from the same *targets* clustered in the same region. K-means clustering is one of the best method that can be used in these kind of situations.

The methods described in Section 3.3.1.1, Section 3.3.1.2 and Section 3.3.1.3, same data is used as input for each one of them. The term data refers to  $n \times 2$  matrix as explained above. For given data, each method tried to find  $k$  and the estimate of  $k$  is compared with exact number of targets,  $n^+$ , that generates the data. The parameter  $x$  and  $y$  in these sections are refer to first column of data and second column of data respectively.

### 3.3.1.1 Exponential Fit

The first method tested for K-means is exponential fit approach which is also used in [30]. In this approach, the defected curve, seen in Figure 3.2, is approximated by an exponential function, shown in Equation (3.2).

$$P = P_1 + P_2 \cdot e^{-x/P_3} \quad (3.2)$$

$$Err = \sum_{i=1}^n \left( y - [P_1 + P_2 \cdot e^{-x/P_3}] \right)^2 \quad (3.3)$$

---

<sup>2</sup>Same equation is used by simulation to generate Detections from tracks, not from targets.

The main goal in exponential fit is to generate a function, formulated as Equation (3.2), that fits the defected curve resulting from the data. Since a perfect fit, a function formulated as Equation (3.2) with margin of error zero meters away from the defected curve, is not feasible to calculate, an error function is needed in order to accept the result of exponential fit function. The error function is formulated as Equation (3.3).

The exponential function, Equation (3.2), takes  $P \in \mathbb{R}^3$  and  $x$  as input, and the error function, Equation (3.3), takes  $P \in \mathbb{R}^3$ ,  $x$  and  $y$  as input. The parameters estimated for exponential fit are evaluated with Equation (3.3) and when they fall within a certain threshold according to Equation (3.3), the result is accepted as a valid solution. The initial guess for  $P$  is not very crucial since  $P$  updates iteratively and converges closer to the exact values in time. It should be kept in mind that Equation (3.3) is used to determine when to accept current  $P$  valid or not. Since  $P$  converges closer to exact  $P$  parameters that fits perfectly to resulting curve from data, an error function is needed for finalizing the computation of  $P$ .

After exponential fit function, formulated as Equation (3.2), with appropriate  $P$  parameter is generated, best- $k$  can be estimated. According to function,  $x$  and  $y$  parameters can be formulated. The  $y$  dimension decreases in each time step (see Figure 3.2), and 90% reduction from first  $y$  parameter is accepted as best  $k$ . Here  $y$  parameter is used as a reference point and the  $x$  component of  $y$  parameter with 90% reduction is returned as best- $k$ .

### 3.3.1.2 Polynomial Fit

Polynomial fit is similar to the method described in Section 3.3.1.1. In polynomial fit, the polynomial fit with degree one, formulated as Equation (3.4), is used to formulate the defected curve resulted from the data. The polynomial fit is used for similar purposes in [30]. In this section, the  $P \in \mathbb{R}^2$ . The  $x$  is input where  $F(x)$  should equal to  $y$  for corresponding  $x$  and  $y$  parameters. The polynomial fit tries to find the best  $P$  parameters that satisfies these results.

$$f(x) = P_1 \cdot x + P_2 \quad (3.4)$$

The Equation (3.4) can be visualized as a straight line which cannot fit the shape depicted in Figure 3.2. The polynomial fit is not used with  $x$  and  $y$  parameters of data, instead  $\log(x)$  is used as  $x$  parameter while  $y$  parameter left untouched. When the  $\log(x)$  against  $y$  parameter is plotted, a similar figure of Figure 3.3 can be seen. It is quite clear a straight line that can fit Figure 3.3 is not very hard to calculate. It should not be forgotten that no perfect  $P$  parameters

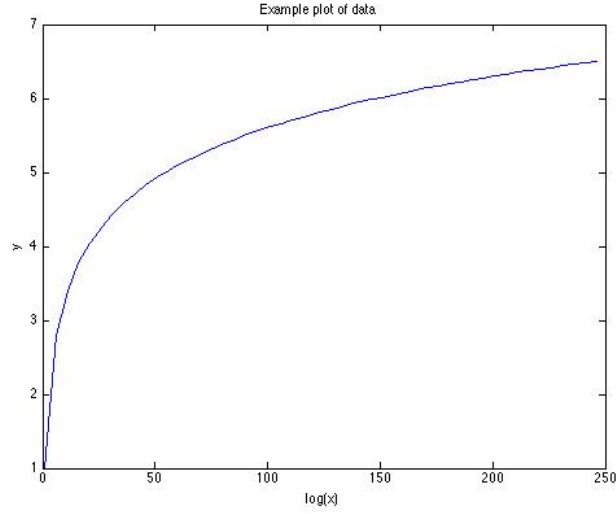


Figure 3.3: An example graph for polynomial fit. It should be noted that the graph consists of  $\log(x)$  and  $y$  parameters of the data in Section 3.3.1.1.

for Equation (3.4) need to be founded, instead an error function is used in order to accept  $P$  parameters with a certain error.

The polynomial fit is the least unlikely method to find a perfect function that can fit Figure 3.2. But the advantage of polynomial fit is its computational requirements. Amongst the three methods for best- $k$  selection, polynomial fit with degree one is the least demanding in terms of computation.

After  $P$  parameters for Equation (3.4) is generated, best- $k$  can be estimated. According to the current knowledge of polynomial fit, the  $x$  and  $y$  parameters can be formulated with Equation (3.4) and best  $k$  can be found with ease. As explained Section 3.3.1.1, the  $y$  parameters decrease in each time step (see Figure 3.2). 90% reduction from first  $y$  parameter is accepted as best  $k$ . Different from the Section 3.3.1.1, instead of  $x$  component of  $y$  parameters with %90 reduction is returned as best  $k$ , the  $e^x$  is returned as best  $k$ . The reason behind this conversion is about the generation of  $x$  data for polynomial fit. In this section  $x$  data is the  $\log(x)$  of actual data, so  $e^x$  provides us the exact  $x$  parameter provided in the first place.

### 3.3.1.3 L-method

There exists different methodologies for finding the best  $k$ . Different methodologies investigated for this purpose. In this paper, the third method used for selecting the best  $k$  is L-method, introduced by [13]. The explanations for L-method and its analyze against similar methods can

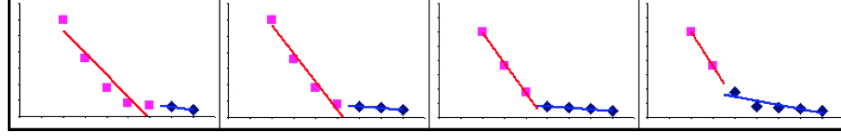


Figure 3.4: Possibilities of line fits for seven data points (from [13])

be found [13]. The explanations are derived from [13].

This method is a candidate method for best  $k$  selection and analyzed against other methods described in Section 3.3.1.2, Section 3.3.1.1. The main goal of L-method is to find the knee point of a curve and since K-means clustering returns a defected curve (see Figure 3.2), L-method can become beneficial for finding the best  $k$ .

L-method tries to find the knee of the curve via fitting two lines to the curve: the first one in the left side of the curve and the second one in the right side of the curve. The intersection point of these lines is accepted as knee of the curve, called the *index*. For any given  $m$  data point of a curve, there exists  $m - 4$  different ways of fitting two curves to the data according to the constraints described in [13].

The L-method tries to fit the lines to the curve for seven data points can be seen in Figure 3.4. It is clear that each line needs at least two data points for line fitting while each point can only belong to one line. Also, the points belong the same line need to be continuous. As a result, it can be inferred that, for given  $m$  data points, the L-method tries to assign first  $n$  data points to one line and  $m - n$  to the other line while each line has at least 2 data points. ( $n \geq 2, m - n \geq 2$ ) Computing a perfect line fit to a curve is infeasible, so a modified error function, Equation (3.5), is used for measuring the accuracy of line fits and accepting the results as valid or not.

The L-method uses total root mean square error (RMSE) for measuring the accuracy of line fits. Assuming the left line is depicted by  $L_n$ , where  $L_n$  consists of data points  $1 \dots n$ , and the right line is depicted by  $R_n$ , where  $R_n$  consists of data points  $n + 1 \dots m$ , and  $n$  is the index point as the intersection of two lines, the graph consists of  $m$  data points and  $n$  can vary from 2 to  $m - 3$ . The total RMSE of line fits can be computed as in Equation (3.5), where the goal is to minimize the  $RMSE_n$ . According to [13], L-method works efficiently for 20 or more data points. Since L-method requires at least 2 data points for both lines, without a modification to the current algorithm, it can only be used for data point more than 4.

$$RMSE_n = \left( \frac{n-1}{m-1} \right) RMSE(L_n) + \left( \frac{m-n}{m-1} \right) RMSE(R_n) \quad (3.5)$$

For each  $n$ , where  $n = 2 \dots m-3$ ,  $RMSE_n$  is generated via Equation (3.5). After error parameters for  $n$  is generated, Equation (3.6) which selects the minimum RMSE amongst the all different line combinations where  $n = 2 \dots m-3$ , is used for selecting the best  $k$ . The  $x$  paramater which provides the least error via Equation (3.5) is accepted as best  $k$ .

$$Bestk = \min(\forall(RMSE_i) | i = 2 \dots m-3) \quad (3.6)$$

The L-method, as described in [13], cannot provide accurate results for the length of data less than 20. As a result, for data less than 20, a modified version of L-method is used for the index. This process mainly tries to expand the data without changing the shape of data, instead introducing intermediate points to the data in order to provide enough data points.

#### 3.3.1.4 Modified L-method

In this paper, in order for selecting best  $k$ , different algorithms are analyzed. According to the problems investigate in this paper, the method for selecting best  $k$  should be able to handle length of data more or equals to 2. But, the L-method cannot handle data with length less than 20. In order to compensate for this problem, a modified version of L-method is derived. It should be kept in mind that modified L-method introduced in this paper is not related to [13] and invented for test purposes. [13] cannot be hold responsible for the results derived via modified L-method.

In order to prevent deal with the situation where length of data is less than 20, the modified L-method generates new  $y$  and  $x$  parameters of the data without losing the orientation of the original data. From this point point, the length of data is referred as  $d$  for simplicity. Modified L-method introduces  $d-1$  data points at each interval unless  $d \geq 20$ . In this paper, this process is referred as *data expanding*.

The first step is to generate  $x^*$  and  $y^*$  according to Equation (3.7), from the original data  $x$  and  $y$ . The size of the new  $data^*$ , with  $x^*$  and  $y^*$  has the length  $d^*$ , where  $d^* = 2 \times d - 1$ . After  $x^*$  and  $y^*$  is generated, the even indice are filled via Equation (3.8).

$$x_{(2 \times i - 1)}^* = \sum_{i=1}^d x_{(i)}, y_{(2 \times i - 1)}^* = \sum_{i=1}^d y_{(i)} \quad (3.7)$$

$$x_{(2 \times i)}^* = \sum_{i=1}^d \frac{x_{(i)} + x_{(i+1)}}{2}, y_{(2 \times i)}^* = \sum_{i=1}^d \frac{y_{(i)} + y_{(i+1)}}{2} \quad (3.8)$$

After  $data^*$  is generated via Equation (3.7) and Equation (3.8), and  $d^* \geq 20$ , L-method is applied for  $data^*$ . If  $d^* \not\geq 20$ , then data expanding method explained above is repeated for  $data^*$  until  $d^* \geq 20$ .

After best k is estimated via L-method for  $data^*$  is executed, the corresponding  $x^*$  that minimizes Equation (3.5) is acquired. But the real  $x$  is found by comparing  $x^*$  of  $data^*$  against  $x$  of data. The  $x$  closest to the  $x^*$  is accepted as best k estimate for L-method.

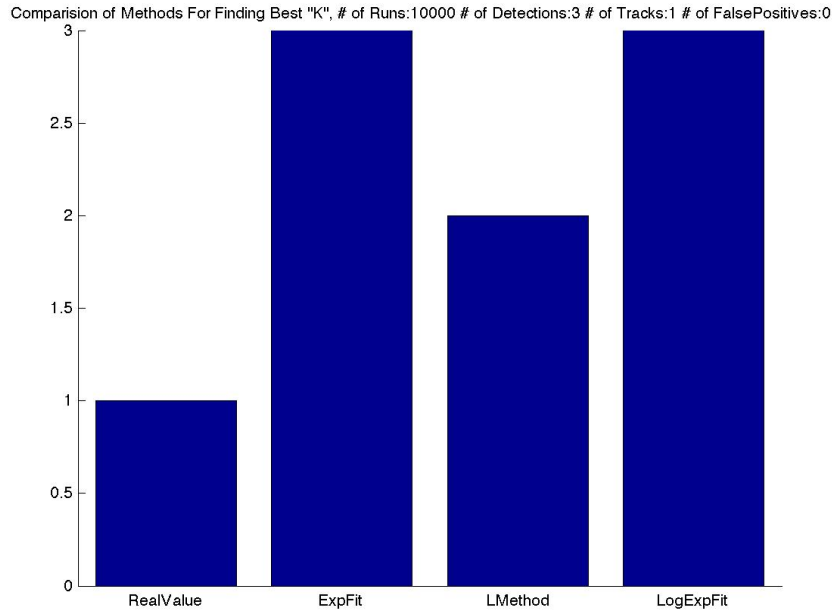
This approach for length of data less than 20 does not guarantee accurate knee estimation, but neither do the other methods. The results of all methods are explained in Section 3.3.1.5

### 3.3.1.5 Evaluation of Clustering Approaches

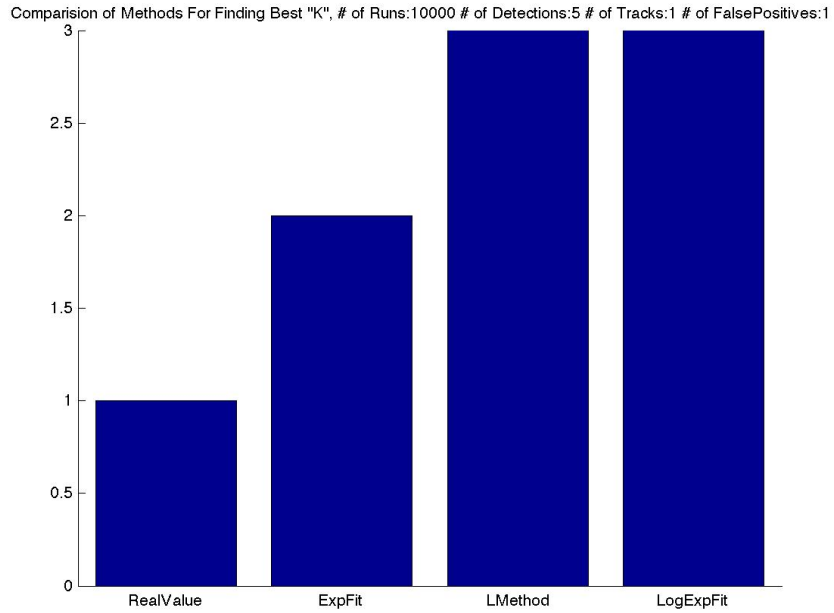
All these methods described in Section 3.3.1.1, Section 3.3.1.2 and Section 3.3.1.3, are tested against each other for a varying test cases with 10000 runs each. In each test case, as explained above, the exact number of detections provided as input where 10% of these detections as false positives, while the remaining detections are generated in the vicinity of pre-generated tracks. The ratio between detection against its originated track is 10% which means each track is likely to generate 10 detections on average while the total number of detections preserved at the end. According to these criteria, the results seen in Figure 3.5, Figure 3.6, Figure 3.7, and Figure 3.8 are generated.

The results in the figures can be summarized as Table 3.5. It can be seen that, for any number of detections, in general, L-method provides the most accurate results, followed by ‘ExpFit’ and ‘LogExpFit’ with the least accurate results. The L-method used in ‘# Detections’ less than 20 is the modified method that is described above. Even if the modified method is not guaranteed to work efficiently, it still works with some accuracy.

It should be noted that both methods described in Section 3.3.1.1 and Section 3.3.1.2 find the best k via 90% reduction in y parameters. This also means that same methods use 10% height

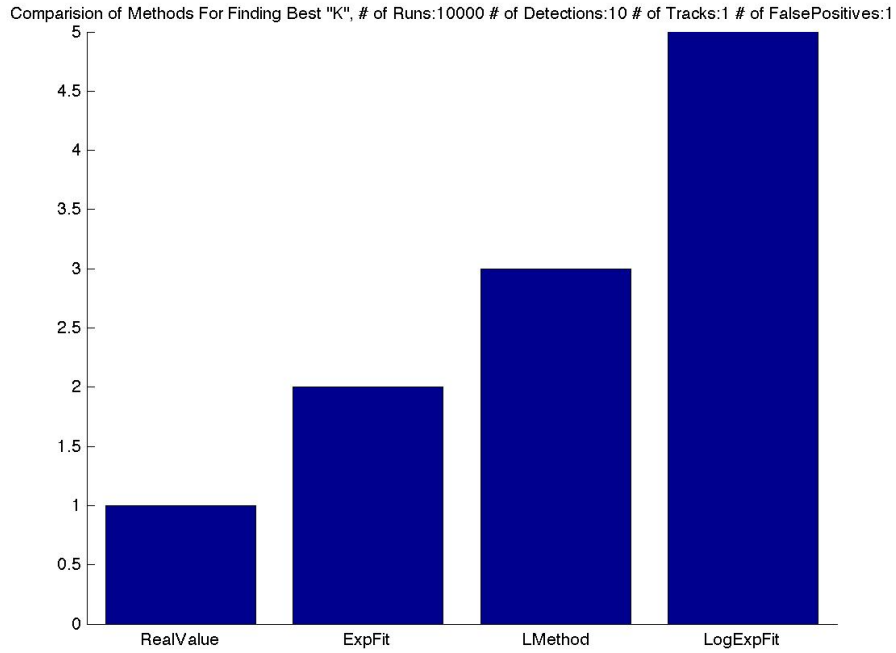


(a) Scenario: 1 Track originates 3 Detections with 0 False Positives

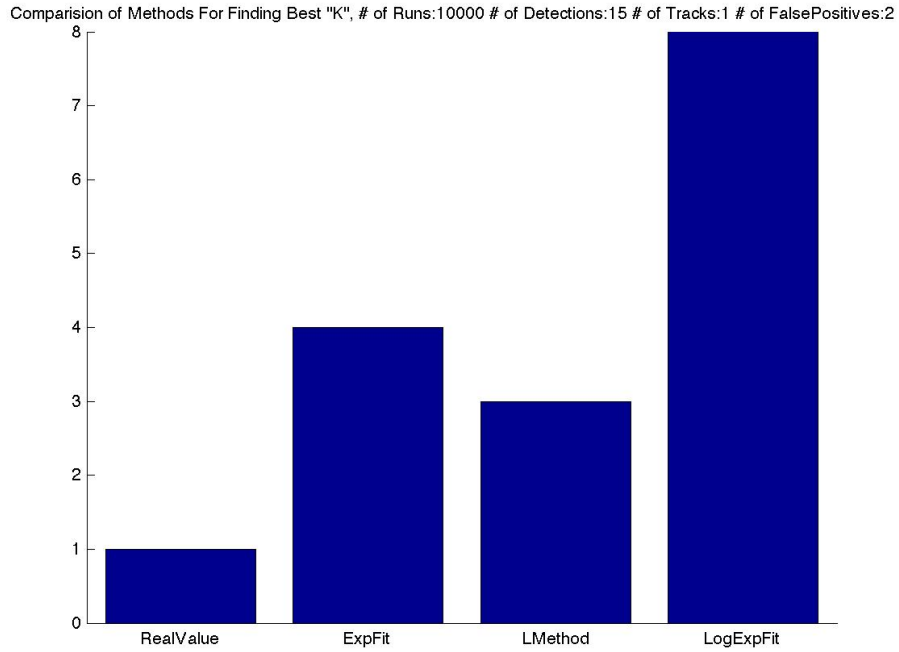


(b) Scenario: 1 Track originates 5 Detections with 1 False Positives

Figure 3.5: Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. RealValue shows the exact number of tracks, ‘real k’, that originates the detections. Methods that finds closer results to ‘real k’ are more accurate. For each scenario, 10000 Monte-Carlo runs established.

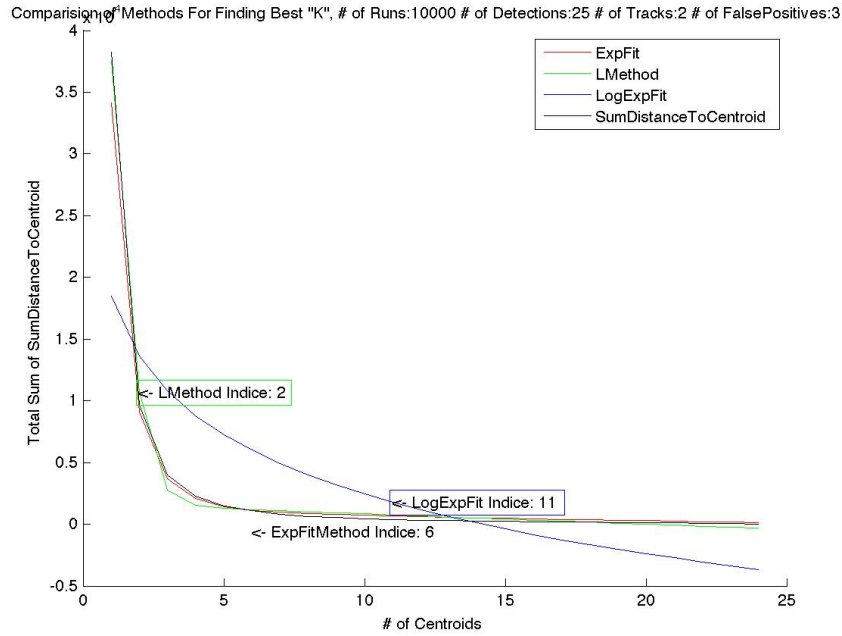


(a) Scenario: 1 Track originates 10 Detections with 1 False Positives

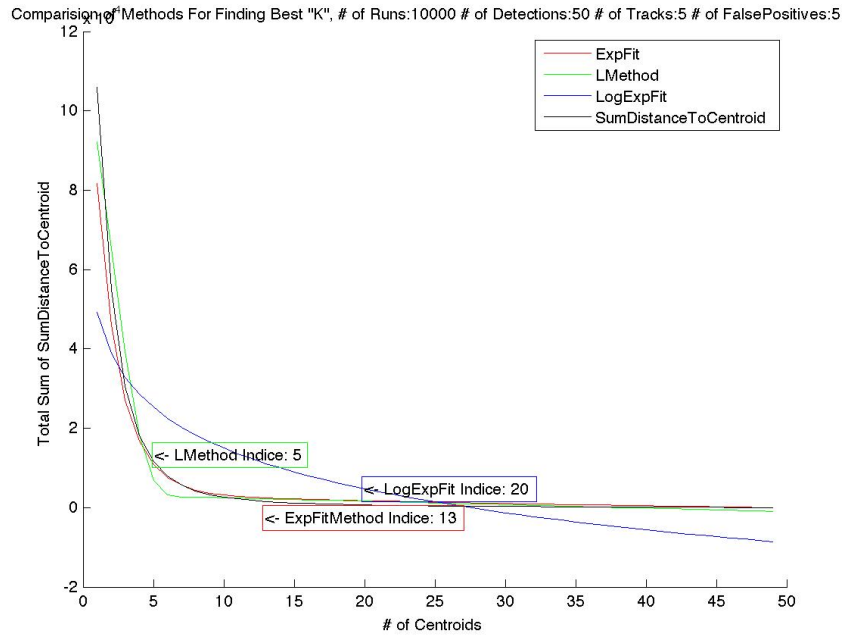


(b) Scenario: 1 Track originates 15 Detections with 2 False Positives

Figure 3.6: Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. RealValue shows the exact number of tracks, ‘real k’, that originates the detections. Methods that finds closer results to ‘real k’ are more accurate. For each scenario, 10000 Monte-Carlo runs established.

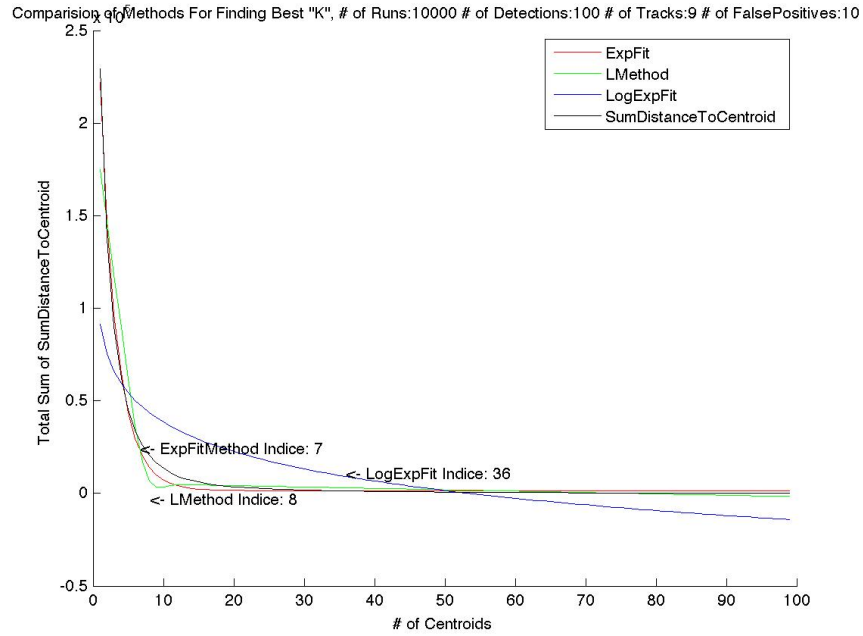


(a) Scenario: 2 Tracks originate total of 25 Detections with total of 3 False Positives

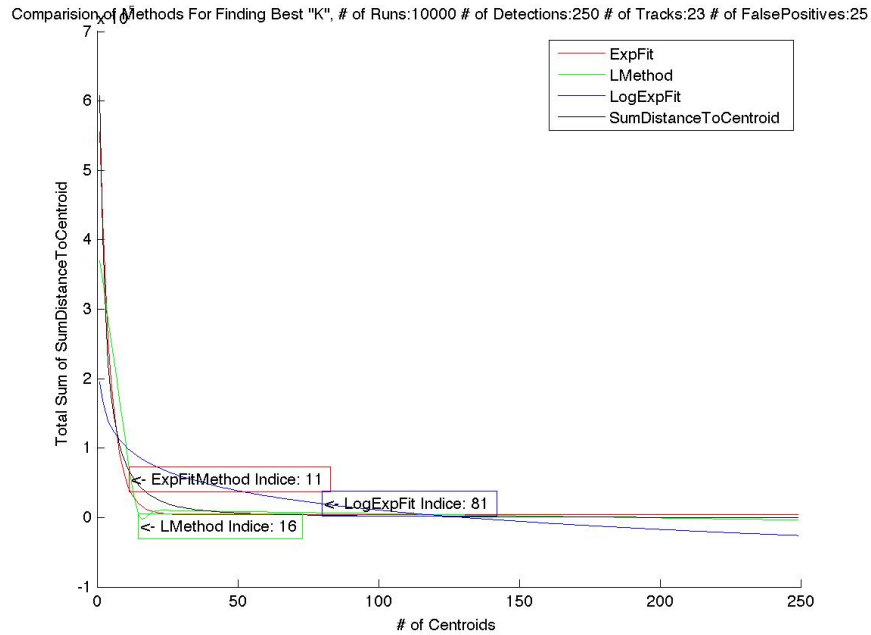


(b) Scenario: 5 Tracks originate total of 50 Detections with total of 5 False Positives

Figure 3.7: Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. ‘SumDistanceToCentroid’ shows the exact curve of the data. Methods that find closer results to ‘# of Tracks’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. The location of the index boxes that shows the ‘Best-K’ the associated method is found, is slightly shifted for readability purposes.



(a) Scenario: 9 Tracks originate total of 100 Detections with total of 10 False Positives



(b) Scenario: 23 Tracks originate total of 250 Detections with total of 25 False Positives

Figure 3.8: Test results for finding Best-K. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3 and LogExpFit is described in Section 3.3.1.2. ‘SumDistanceToCentroid’ shows the exact curve of the data. Methods that find closer results to ‘# of Tracks’ are more accurate. For each scenario, 10000 Monte-Carlo runs established. The location of the index boxes that shows the ‘Best-K’ the associated method is found, is slightly shifted for readability purposes.

Real Parameters			Simulation Results		
# Detections	# FP	# Tracks	L Method	ExpFit	LogExpFit
3	0	1	2	3	3
5	1	1	3	2	3
10	1	1	3	2	5
15	2	1	3	4	8
25	3	2	2	6	11
50	5	5	5	13	20
100	10	9	8	7	36
250	25	23	16	11	81

Table 3.5: The results of selecting best K algorithms. ExpFit is described in Section 3.3.1.1, LMethod is described in Section 3.3.1.3, and LogExpFit is described in Section 3.3.1.2. The red numbers represent the closest estimate in terms of the real number of tracks.

# Detections	# FP	# Tracks	Height of the index
5	1	1	0.0182
10	1	1	0.0421
15	2	1	0.2136
25	3	2	0.1042
50	5	5	0.0743
100	10	9	0.0585
250	25	23	0.0352

Table 3.6: For each scenario, the parameters of the scenario are described under the ‘Real Parameters’ column, and the distance from the  $x$  dimension of the ‘real index’, or ‘Best- $K$ ’, or ‘Number of Tracks’, is described under the ‘Height of the index’ column. The real index distance is normalized for each scenario. The methods described in Section 3.3.1.1 and Section 3.3.1.2 are using %0.9 reduction ratio from the maximum distance which is same as 0.1000 ‘Height of the index’ in this table. The normalization is done for each scenario itself. It should be noted that normalization is not done after all heights for all scenarios are computed. Each normalization of ‘Height of the index’ is independent from any other scenario.

of  $y$  parameters as critical value. 10% is not a guaranteed ratio that can find the best  $k$ . In order to see what is the real height of  $y$  for real value, Table 3.6 is generated from the same data that generates the Table 3.5. For each given scenario, the exact height of  $y$  parameter, called ‘Height of the index’, against the real number of targets, called ‘# Tracks’, is shown in Table 3.6. It is quite clear that even with a fixed ratio of false positives, the height varies 0.0182...0.2136. The methods described in Section 3.3.1.1 and Section 3.3.1.2 used 0.1000 as height but it is concluded that there exists no way to find a fixed height parameter. So the results from the methods described in Section 3.3.1.1 and Section 3.3.1.2 cannot be further improved.

The due the performance of L-method and inability to further improve methods described in Section 3.3.1.1 and Section 3.3.1.2, L-method, Section 3.3.1.3, is accepted as the main and only method for finding the knee of the curve after K-means clustering is called. For any given number of Detections, L-method is used. If the length of the data is less than 20, modified version of L-method, Section 3.3.1.4, is used as described above.

In this paper, K-means method is used for track generation. The analyzes in Section 3.3.1.5 shows that K-means algorithm works best with L-method according to the constraints introduced. According to the information below, after agent acquires all of its Detections, K-means method is called, as described in Section 3.3.1. K-means acquires Detections and generates tracks from these Detections. After tracks are generated, covariance intersection method described in Section 3.3.2 is called in order to generate the belief matrix of error, called  $\Sigma_t$ , of each track. Section 3.3.2 generates the intersection of the  $\Sigma_d$  of the Detections that generates the associated track. The  $\Sigma_t$  of the tracks are  $\in \mathbb{R}^{[3,3]}$  while location of the tracks, called  $\bar{t}$ , are  $\in \mathbb{R}^6$ .

### 3.3.2 Covariance Intersection

Covariance intersection method, described in [10], fuse the state information of multiple Gaussian state estimations. Covariance intersection method tries to fuse the all covariance information in order to generate the intersection of all covariance knowledge. In this paper, the method described by [10] is used.

Assume there exists  $N$  state estimates with mean  $\hat{x}_i$  and covariance  $P_i$  of state  $i$ , with weighting factor  $w_i$ , the fused mean of states is denoted  $\hat{x}_0$  with covariance  $P_0$ . According to [10], the fused covariance of multiple Gaussian state estimations can be computed via Equation (3.9) and the mean of a Gaussian state estimation can be computed via Equation (3.10). It should be noted that without computing the covariance  $P_0$ , the mean of states,  $\hat{x}_0$  cannot be computed.

$$P_0^{-1} = \sum_{n=1}^N w_n P_n^{-1} \quad (3.9)$$

$$P_0^{-1} \hat{x}_0 = \sum_{n=1}^N w_n P_n^{-1} * \hat{x}_n \quad (3.10)$$

Though [10] does not introduce a specific method to generate weighting factor  $w_i$ , the sum of all

$w_i$  should add up to one. In the presented work,  $w_i$  is generated as the trace of the covariance,  $P_i$ , of each state  $i$  and normalized afterwards. So, weighting factors satisfy the criterion introduce by [10].

In this paper, covariance intersection method is applied in Algorithm-9. According to the information flow of the agent, prior to covariance intersection, agent has the tracks generated and  $\hat{t}$  is assigned for each track. Also, the relationship between Detections and tracks is known. The covariance intersection is called for the Detections that the track is originated from. For all Detections, according to Equation (3.9) and Equation (3.10), the  $P_i$  equals to the  $\Sigma_d$  and  $\hat{x}_i$  is equals to  $\hat{d}$ .

---

**Algorithm 9** Agent

---

```

31: procedure COVARIANCEASSESSMENT(Detections)
32:   for t=1 to sizeof (tracks) do
33:     DetectGrp  $\leftarrow \forall$ Detections generates Tracks(t)
34:     for i=1 to sizeof (DetectGrp) do
35:        $P_i \leftarrow \Sigma_{d_i}$ 
36:        $w_i \leftarrow \text{normalize}(\text{trace}(P_i))$ 
37:        $P_0^{-1} \leftarrow w_i * P_i^{-1}$ 
38:     end for
39:      $P_0 \leftarrow \text{normalize}(P_0)$ 
40:     for i=1 to sizeof (DetectGrp) do
41:        $\hat{x}_i \leftarrow \hat{d}_i$ 
42:        $\hat{x}_0 \leftarrow w_i * P_i * \hat{x}_i$ 
43:     end for
44:      $\hat{x}_0 \leftarrow P_0 * \hat{x}_0$ 
45:      $\hat{t} \leftarrow \hat{x}_0$ 
46:      $\Sigma_t \leftarrow P_0$ 
47:   end for
48: end procedure

```

---

The covariance intersection, which generates the  $\Sigma_t$  for tracks, is generated by Algorithm 9. This algorithm first generates a group of detections, called ‘DetectGrp’ that consists of the Detections where the track is originated from, and use this ‘DetectGrp’ to extract  $\hat{d}$  and  $\Sigma_d$  and generate  $\hat{x}_i$  and  $P_i$  for each detection.<sup>3</sup>

After all  $w_i$ ,  $\hat{x}_i$  and  $P_i$  are generated for given ‘DetectGrp’, the  $\Sigma_t$  is generated according to the

---

<sup>3</sup>The Detections that generates the same track becomes state estimation in here. Both state estimation and detection refers the same parameter.

Equation (3.10). This process is depicted in Algorithm 9.

It should be noted that the  $\Sigma_t$  of a track is the belief matrix of noise for each state dimension of the track that is generated as the intersection of each track's relevant detections'  $\Sigma_d$ . The  $\bar{t}$  of a tracks is its cartesian coordinates. Even if  $\bar{d}$  and  $\Sigma_d$  for detections and  $\bar{t}$  and  $\Sigma_t$  for tracks provides the same information, they are generated with different approaches.

Covariance Intersection method is used to compute the aggregated uncertainty from the clustered detections. The collective output of these steps are tracks, which possess both a mean value and covariance matrix, representing the fused sensor measurements and uncertainties. It should be noted that both K-means method and covariance intersection generates mean values of given data, which equals to  $\bar{t}$  in this paper, but the K-means results are assigned to the  $\bar{t}$ . The results of covariance intersection is not used for this purpose.

### 3.4 Track Association

After agent acquires its all detections, it needs to analyze them for further improvements. The main goal in this process is to use KF to further improve the target estimates. In order to use KF, the parameters of the KF needs to be carried over throughout the time for the associated tracks. All of this process is handled in this section.

All of this process is handled as seen in Algorithm-10. Except for the first time an agent is called, agent tries to associate tracks at time  $t$  with its tracks at time  $t - 1$ . In order to associate tracks, agent finds the likelihood of all possible associations via 'JPDA' function, which is detailed in Section 3.4.1. JPDA generates  $\beta$  and  $\beta_0$  probabilities which depicts the all possible likelihood parameters for all tracks.

The results of the 'JPDA' is further analyzed by 'Munkres' function, as explained in Section 3.4.2. The 'Munkres' function provides the association of tracks according to the given likelihood parameters. At the end of Section 3.4.2, each track at time  $t$  is associated the tracks at time  $t - 1$ . If a track has no associated track in preceding time step, it is accepted as a new track.

For each track that has an associated track in preceding time step, the KF parameters are carried over to the current track. This is crucial for KF in order to work efficiently which is detailed in Section 3.5. For any other track that has no associated track in preceding time step, a new KF is initialized. After each track acquired its own KF parameters, all of the tracks parameters are

---

**Algorithm 10** Agent

---

```
49: procedure PARSETRACKS(tracks)
50:   if  $History(t) \neq \emptyset$  then
51:      $[\beta, \beta_0] \leftarrow JPDA(History(t).Tracks, tracks)$ 
52:      $Indice \leftarrow \text{Munkres}(\beta \cup \beta_0)$ 
53:      $NewTrackIndice \leftarrow Tracks \notin Indice$ 
54:     if  $Tracks \in Indice$  then
55:        $Tracks.KF \leftarrow History.Tracks.KF$ 
56:     end if
57:     if  $Tracks \in NewTrackIndice$  then
58:        $Tracks.KF \leftarrow KalmanFiltering()$ 
59:     end if
60:   end if
61:    $\bar{t}, \Sigma_t \leftarrow Tracks.KFUpdate()$ 
62: end procedure
```

---

updated according to KF, which is detailed in Section 3.5. By applying KF to each tracks,  $\bar{t}$  and  $\Sigma_t$  parameters are further improved.

All of the methods used in this section are further detailed by the following sections.

### 3.4.1 Joint Probabilistic Data Association

One of the key issues in MTT systems is the association of observations with their relevant tracks at each time step in order to identify the tracks. The association problem—means associating each observation with its relevant target—is investigated by [16–21, 31] and many more. In order to solve this problem, one of the well-known method is known as JPDA. JPDA method provides accurate results when the number of tracks in the system is known. But when the opposite situation is the case, the accuracy of observations association with tracks diminishes. The JPDA method provides optimal results with increased computational requirements as the complexity arises. In order to deal with computational requirements of JPDA, there exist different versions for similar MTT situations.

One of the different version of JPDA algorithm is known as Suboptimal JPDA that requires less computational requirements but provides suboptimal solutions with heuristic approach instead of an optimal solution. Suboptimal JPDA's accuracy diminishes when the False Positive (FP) observations introduced to the system. One of the another method, introduced by [21], can provide better results with same complexity when FP observations are introduced to the system.

This method is known as the Augmented Suboptimal JPDA (ASJPDA) algorithm and described in [21] with its comparisons against other JPDA methods.

The ASJPDA, which is used in this paper, is introduced by [21]. The explanations below for ASJPDA is derived from [21]. The detailed explanation for ASJPDA and its complexity analyze against similar algorithms can be found in [21].

ASJPDA consists of 7 steps and generates two different tables at the end in which one table provides association probabilities of observations with their relevant targets where other table provides the probability of observations of not having any relevant target. ASJPDA does not provide one to one associations of observations against targets. There exists different methods in order to deal with this problem which is described in Section 3.4.2.

According to the information flow of an agent, ASJPDA is called when an agent has parsed its detections, generated its tracks, but not called KF for any track. The agent tries to associate its total number of  $z^*$  tracks at time  $t-1$ , depicted as  $Tracks_{z^*}^{t-1}$ , against total number of  $z$  tracks at time  $t$ , depicted as  $Tracks_z^t$ . In this case,  $Tracks_{z^*}^{t-1}$  become observations and  $Tracks_z^t$  become targets. It should be kept mind that both  $Tracks_{z^*}^{t-1}$  and  $Tracks_z^t$  are observations of the agent for different time steps. Agents have no knowledge of targets.

[21], in the paper, uses the terms tracks and *hits* which corresponds  $Tracks_{z^*}^{t-1}$  and  $Tracks_z^t$  respectively in this case. The term tracks and *hits* will be used from this point on.

The main reason for calling ASJPDA is to associate tracks with their respective tracks in the previous state. This process allows both to identify tracks and improve their estimates via KF, which needs to carry out its parameters at each time step, as described in Section 3.5.

Since the tracks are  $Tracks_{z^*}^{t-1}$ , their current state estimates needs to be computed. This process is handled according to state dynamic matrix in Equation (2.4) with the formula Equation (2.5) prior to ASJPDA algorithm but does not stored as new  $Tracks_{z^*}^{t-1}$  for the agent. Only their predicted state of time  $t$  is send as tracks for ASJPDA.

The first step in ASJPDA is to determine the validated *hits* of each track  $t$  and form  $A_t$ , shown in Equation (3.11). The second step is to do the same process for *hits*, which means to determine the validated targets of each *hit*  $j$  and form  $C_j$ , shown in Equation (3.12).

$$A_t = \sum_{j \in A_t} Hits_j \quad (3.11)$$

$$C_j = \sum_{t \in C_j} Tracks_t \quad (3.12)$$

The validation process of *hits* against tracks or vice versa is handled by Bhattacharyya distance, described in Section 3.4.1.1. Bhattacharyya distance, used with a *hit* and a track, is used to determine the similarity of two discrete or continuous probability distributions. Since each *hit* and each track in JPDA is a continuous probability distribution, Bhattacharyya distance is used to compute the extension gate of a track or *hit*. When the Bhattacharyya distance is computed for given *hit* and track, the predetermined threshold value is used to determine whether or not to accept the *hit* in the extension gate of the given track or vice versa. If the Bhattacharyya distance is less than predetermined threshold value, the *hit* is accepted in the extension gate of the track.

After the first and second steps in ASJPDA are executed, for each track  $t$ , the tracks that may fall in the extension gate of validated hits of given track  $t$  is computed and stored as  $L_t$ , as the third step. The  $L_t$  excludes the associated track  $t$  while including all other tracks that are in the extension gate of its validated hits. This can be formulated as Equation (3.13).

$$L_t = \bigcup_{j \in A_t} C_j - \{t\} \quad (3.13)$$

After  $L_t$  is generated, cardinality of the largest measurement index is computed and stored as  $c_t$  as in Equation (3.14).

$$L_t = \max_{u \in L_t} |A_u| \quad (3.14)$$

According to the number of hits in the extension gate of track  $t$ , the association probabilities of track with its validated hits is computed according to Equation (3.15). In this paper, the  $\Omega_t$  or  $\Theta_{ty}$  parameters are used instead of  $P_t$  in order to prevent ambiguity.

$$P_t = \begin{cases} \Omega_t = \Lambda_{tj} & \text{if } |A_t| = 1 \\ \Theta_{ty} = \max_{k \in A_t, k \neq j} \Lambda_{tk} & \text{if } |A_t| > 1 \end{cases} \quad (3.15)$$

It is quite clear that Equation (3.15) uses  $\Lambda_{th}$  parameter in order to generate results both for  $\Omega_t$  and  $\Theta_{ty}$ .<sup>4</sup>  $\Lambda_{th}$  is the Gaussian probability of the likelihood value of given track  $t$  against given *hit*  $h$ . The process of computing the likelihood of track against a hit is described in Section 3.4.1.2.

For each track  $t$  and its validated measurement hit  $j$ , the  $Dtj$  is computed as Equation (3.16) in accordance with Equation (3.17), Equation (3.19) and Equation (3.18).

$$D_{tj} = \begin{cases} \Lambda_{tj}(\sum_{u \in L_t} N_{tuj} + b) & \text{if } c_t \geq 2 \\ \Lambda_{tj} + \sum_{u \in L_t} Q_{uj} & \text{if } c_t < 2 \end{cases} \quad (3.16)$$

$$N_{tuj} = \begin{cases} \Theta_{uj} & \text{if } |A_u| \geq 2 \text{ and } j \in A_u \\ \Omega_u & \text{if } |A_u| = 1 \text{ and } j \notin A_u \\ \Lambda_{tj} & \text{otherwise} \end{cases} \quad (3.17)$$

$$Q_{uj} = \begin{cases} \Omega_u & \text{if } j \notin A_u \\ 0 & \text{otherwise} \end{cases} \quad (3.18)$$

According to Equation (3.19), it is required to decide on clutter spatial density and target detection probability, which is represented as  $\lambda$  and  $P_D$  in Equation (3.19) respectively. These two parameters are predetermined and need to be evaluated wisely. If they don't evaluated wisely, they may hamper to overall efficiency of ASJPDA.

$$b = \frac{\lambda(1 - P_D)}{P_D} \quad (3.19)$$

After all the parameters explained above is computed, for each track  $t$ ,  $B_t$  needs to be computed as in Equation (3.20). It should be noted that Equation (3.19) has impact on both Equation (3.16) and Equation (3.20).

---

<sup>4</sup>Here,  $\Lambda_{th}$  represents both  $\Lambda_{tj}$  and  $\Lambda_{tk}$  of Equation (3.15).

$$B_t = \begin{cases} b(b + \sum_{u \in L_t} \sum_{j \in A_t} \Lambda_{uj}) & \text{if } c_t \geq 2 \\ b + \sum_{u \in L_t} \sum_{j \in A_t} \Lambda_{uj} & \text{otherwise} \end{cases} \quad (3.20)$$

Finally, for each track  $t$ , the probabilities of a tracks association with its validated hits  $j$  are computed in Equation (3.21) while a track is not associated to any hit is computed in Equation (3.22).

$$\beta_{tj} = \frac{D_{tj}}{B_t + \sum_{k \in A_t} D_{tk}} \quad (3.21)$$

$$\beta_{0t} = \frac{B_t}{B_t + \sum_{k \in A_t} D_{tk}} \quad (3.22)$$

For each row of  $\beta$  and  $\beta_0$ , the sum of each row adds up to 1 as can be seen in Equation (3.23).

$$\sum_{j=0}^m B_{tj} = 1 \quad (3.23)$$

It should be keep in mind that ASJPDA provides probabilities of a track association with its validated hits with a probability that a track is not associated with any hit. These probabilities are returned as  $\beta$  and  $\beta_0$  respectively. ASJPDA never provides insight of how to accept a hit in the extension gate of a track, how to compute association probability of a track and a hit, and how to associate each track with a hit or accept a track is not associated with any hits. The first two problems mentioned here is solved according to Section 3.4.1.1 and Section 3.4.1.2. The last problem, which is all about how to use  $\beta$  and  $\beta_0$  parameters at then end, is solved in Section 3.4.2.

In design perspective, after the first time step of ASJPDA, for each track, if the track has no validated hit, it is accepted that ASJPDA is not needed since no track has no hits in their extension gate. As a result, ASJPDA returns  $\beta$  as 0 and  $\beta_0$  as 1 for each track  $t$  in Equation (3.21) and Equation (3.22) without making any more computation.

### 3.4.1.1 Bhattacharyya Distance

The Bhattacharyya distance, which is used in this paper, is introduced by [32]. The explanations below for Bhattacharyya distance is derived from [32] and the detailed explanation can be found in [32]. There exists applications that uses Bhattacharyya distance such as [33]. It is a well studied method that used in wide variety of applications.

For any given continues probability distribution, the similarity of given two distribution, or the overlap of two statistical distribution, can be computed via Bhattacharyya distance. In this paper, each distribution used in Bhattacharyya distance, which is shown in Equation (3.24), is a Gaussian distribution with mean vector  $\chi$ , as in Equation (3.25) and positive definite covariance matrix  $\Sigma$ , as in Equation (3.26).

$$x = N_n(\chi, \Sigma) \quad (3.24)$$

$$x_n = [X_1, X_2, \dots, X_n] \quad (3.25)$$

$$\Sigma_{n,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \quad (3.26)$$

When both distributions are Gaussian, the Bhattacharyya distance of two distribution becomes Equation (3.27) and Equation (3.28)

$$\Sigma = \frac{\Sigma_1 + \Sigma_2}{2} \quad (3.27)$$

$$Dist_B = \frac{1}{8}(\chi_1 - \chi_2)^T \Sigma^{-1}(\chi_1 - \chi_2) + \frac{1}{2} \ln \left( \frac{\det \Sigma}{\sqrt{\det \Sigma_1 \det \Sigma_2}} \right) \quad (3.28)$$

Since the probability distributions for Bhattacharyya distance are acquired from the JPDA algorithm, it can be deduced that both distributions, which are the hits and the tracks described

in Section 3.4.1, are Gaussian. When agent workflow is traced back, it can be concluded that these hits and tracks are tracks of the agent at time  $t$  and  $t - 1$  respectively. It is known that these tracks are generated from Detections of the agent as described at Section 3.3.1. Each detection is acquired from target according to Equation (2.1) and Equation (2.6). As a result, it can be concluded that both hits and tracks have same properties in which both has positive definite covariance matrix, which becomes  $\Sigma$  of Equation (3.27) and Equation (3.28), and mean, which becomes  $\chi$  of Equation (3.28).

When hits and tracks of the JPDA are investigated, it can be seen that both hits and tracks have the mean  $\hat{t} \in \mathbb{R}^6$ , used as  $\chi$  in Equation (3.29) and Equation (3.30) respectively. Since  $\chi_{Hits}$  consists of 0 parameters in its last 3 elements and  $\Sigma_t$  of both tracks and hits, which are computed according to Section 3.3.2, are  $\in \mathbb{R}^{3 \times 3}$ , optimum way to compute Bhattacharyya distance is to ignore last 3 elements of  $\chi$ . This process is also required for dimension compliance of  $\chi$  with  $\Sigma$ .

$$\chi_{Hits} = [Val_1, Val_2, Val_3, 0, 0, 0] \quad (3.29)$$

$$\chi_{Tracks} = [Val_1, Val_2, Val_3, Val_4, Val_5, Val_6] \quad (3.30)$$

When a hit and a track is used as an input for Equation (3.28), it uses both track's and hit's  $h_{att}$  as  $\chi$  and their  $\Sigma_t$  as  $\Sigma$  parameter and returns the similarity value of given track and hit. Since the  $\Sigma$  is positive definite matrix, as it is built in that way starting from initialization phase of the agents, both  $\Sigma^{-1}$  and  $\det \Sigma$  can calculable and  $\in \mathbb{R}^{[3,3]}$ . As a result, it can be concluded that the  $Dist_B$  of Equation (3.28) is  $\in \mathbb{R}^1$ .

#### 3.4.1.2 Probability Density Function

The multivariate Gaussian distribution is the extended case of one dimensional normal distribution. A normal distribution of variable  $x$  can be described as Equation (3.24). The multi-dimensional Gaussian distribution, which is the extended case for one dimensional Gaussian distribution, is investigated by [34]. The detailed explanation of Equation (3.31), which is used in this paper, can be found in [34].

Since the multivariate normal distribution of a  $n$ -dimensional random vector  $x$  can be written as Equation (3.24) with mean vector  $\chi$  and covariance matrix  $\Sigma$  as seen in Equation (3.25) and

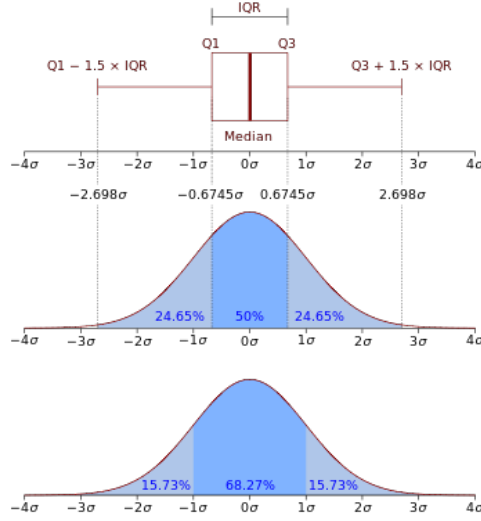


Figure 3.9: PDF distribution of vector  $x$  with  $\chi = 0$ ,  $\Sigma = \sigma$  where the probability of a point belongs to the vector  $x$  is computed as the area under the curve. (Figure taken from [35].)

Equation (3.26) respectively, the probability of any given point that may belong to vector  $x$  can be computed as Equation (3.31). The Gaussian distribution of any  $x$  can be visualized as in Figure 3.9. Both the left and the right hand of the Figure 3.9 goes to  $\infty$ . The area under the curve represents the probability of a point that may belong to  $x$ . It should be noted that the probability depends of  $\chi$  and  $\Sigma$  are parameters of  $x$  and the location of the point, lets call it  $y$  point. As a result, for any point  $y$ , the probability of point  $y$  belongs to Gaussian distribution  $x$ , which is denoted as  $f_X(y)$ , can be computed as Equation (3.31) where  $\chi$  and  $\Sigma$  are parameters of Gaussian distribution  $x$ , and  $y$  is the point with its location for Gaussian distribution  $x$ .

$$f_X(y) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(y - \chi)^T \Sigma^{-1}(y - \chi)\right) \quad (3.31)$$

Equation (3.31) is applicable as long as the  $\Sigma$  is positive-definite, which is the case in this paper as explained in Section 3.4.1.2. If  $\Sigma$  is not positive-definite, there exists other functions to compute the probability which is not needed in this paper. The PDF is used for computing the probability of a hit that may belong to the given track. The  $\Sigma$  parameter is  $\Sigma_t$  of the given track,  $y - \chi$  is computed as the difference of the  $\hat{t}$  of given hit and track. Due to the reasons explained in Section 3.4.1.1, only first 3 dimension of the  $\hat{t}$  parameter of tracks and hits are used for  $y$  computation of Equation (3.31).

It should be kept in mind that, in this paper, Section 3.4.1.1 is used to generate Equation (3.11) and Equation (3.12) while Section 3.4.1.2 is used to generate Equation (3.15). [21] does not define a specific method to compute Equation (3.11), Equation (3.12) and Equation (3.15), but these two methods are used in this paper.

### 3.4.2 Munkres Algorithm

The assignment problem in polynomial time with minimum cost is a problem that is investigated throughout the time. The assignment problem can easily depicted with matrices where each cell of the matrix represents the cost of its associated row and column. For any given problem, if the problem can be represented in a matrix form where each cell is the cost of its associated row and column, the problem can be solved via Munkres algorithm, which is also known as Hungarian algorithm. The original algorithm is designed for square matrices, [36], while it is extend to rectangular matrices via [37].

A simple example can easily portray how to adapt a problem to a matrix form. Lets assume there exists three different users, ‘Jack’, ‘John’ and ‘Jill’ where each one of them can perform ‘Swimming’, ‘Running’ and ‘Jogging’ at gym. The energy cost of each activity for each person can be seen in Equation (3.32). The Munkres algorithm can found the least cost solution as long as the problem is depicted as Equation (3.32).

$$\begin{array}{c} \text{Swimming} \quad \text{Running} \quad \text{Jogging} \\ \text{Jack} \left( \begin{array}{ccc} 1 & 2 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 2 \end{array} \right) \end{array} \quad (3.32)$$

Here, Equation (3.32), Munkres algorithm finds that Jack needs to swim, John needs to run and Jill needs to jog in their exercise for minimum energy cost. As long as the matrix established properly, Munkres algorithm finds the minimum cost.

The extended case for Munkres algorithm for rectangular matrices, which is used in this paper, is introduced by [37]. The explanations below for extended Munkres algorithm is derived from [37]. The detailed explanation of the algorithm and complexity analyze can be found in [37]

Lets assume a problem similar to the example above is analyzed and matrix A is formed where  $r$  is the number of the rows,  $c$  is the number of the columns and  $k$  and  $l$  are computed as

Equation (3.33). The extended Munkres algorithm, which is described below with six steps, is used for association problem of the matrix A. In order to describe the flow of the algorithm these six steps of Munkres algorithm, introduced by [37], used as a reference.

$$k = \min(A) \quad l = \max(A) \quad (3.33)$$

In general, Munkres algorithm keep track of 0 elements of matrix A, which are known as ‘starred zeros’ and stored to matrix Z. When a 0 element of a matrix is found, it is starred in order to keep track of known 0s and stored to Z matrix. In this paper, Z matrix is used for this purpose. The steps of [37] are described below.

Prior to initialing Munkres algorithm,  $k$ ,  $l$ ,  $r$  and  $c$  of matrix A is computed. Then the process starts according to Equation (3.34).

$$Step = \begin{cases} A^+ = \sum_{rr=1}^r \sum_{cc=1}^c A_{rr,cc} = A_{rr,cc} - \min(A_{rr}) \text{ then go to Step 0} & \text{if } r > c \\ \text{go to Step 1} & \text{if } r < c \end{cases} \quad (3.34)$$

- Step 0: For each column of matrix  $A^+$ , generate matrix  $A^{++}$  by subtracting the smallest entry of the column from each entry of the column for each column. Assign  $A^{++}$  to matrix A. Then, go to step 1.
- Step 1: Find a 0, Z, of matrix A. If there is no ‘starred zeros’ neither in its rows nor in its columns, star Z. Iterate this process for each 0 element of matrix A. Then, go to step 2.
- Step 2: Cover every column that contains ‘starred zeros’ in its elements. If  $k$  columns are covered, ‘starred zeros’ are the independent set, the result. Otherwise, go to step 3.
- Step 3: Choose a non covered 0 and prime it, referred as ‘prime zero’. If there exists no ‘starred zero’ in the row of the selected ‘prime zero’, go to step 4. Otherwise cover this row and uncover the column of Z. Iterate this process until all 0s are covered. Then, go to step 5.
- Step 4: The sequence of ‘starred zeros’ and ‘primed zeros’ is alternated as follows: Assume  $Z_0$  represents the ‘prime zeros’,  $Z_1$  represents the ‘starred zeros’ in  $Z_0$ ’s columns, and  $Z_2$  represents the ‘primed zeros’ in  $Z_1$ ’s row. Iterate this process until it stops at ‘prime zeros’,  $Z_{2k}$ , which has no ‘starred zeros’ in its column. When this is achieved,

Unstar each ‘starred zeros’ of the sequence and convert ‘prime zeros’ of the sequence to ‘starred zeros’. Uncover all lines and ‘prime zeros’ and return to step 2.

- Step 5: Assume  $h$  is the smallest non covered element of matrix A. Add  $h$  to every covered row A and subtract  $h$  from every uncovered row A. Without modifying ‘prime zeros’, ‘starred zeros’ and covered lines, go to step 3.

According to the information flow of an agent, the extended Munkres algorithm is used after ASJPDA is executed, prior to KF for each track. ASJPDA provides  $\beta$  matrix and  $\beta_0$  column vector as a result where each member of  $\beta$  or  $\beta_0$  can be utmost 1, according to Equation (3.23).<sup>5</sup> Since there is no guarantee that a square matrix is generated from  $\beta$  and  $\beta_0$ , the process for matrix generation is explained below, the methodology depicted in [37] is required in order solve the association problem for rectangular matrices. It should be kept in mind that after ASJPA, there exists  $n$  number of tracks,  $m$  number of hits and  $n \neq m$  or  $n = m$  can be true.

Munkres algorithm only accepts one matrix and associates one row per column. As a result, in order to allow Munkres algorithm to decide a track has any associated hit or not,  $\beta_0$  column vector is converted to a diagonal matrix as seen in Equation (3.35) and concatenated with  $\beta$ , as seen in Equation (3.36) to generate one matrix where each track is a row and each hit is a column.

$$Matrix_{\beta_0} = diag(\beta_0) = \begin{pmatrix} \beta_{0[1,1]} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_{0[n,n]} \end{pmatrix} \quad (3.35)$$

$$Matrix = (Matrix_{\beta_0} | \beta) = \begin{pmatrix} \beta_{0[1,1]} & \cdots & 0 & \beta_{1,1} & \cdots & \beta_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_{0[n,n]} & \beta_{n,1} & \cdots & \beta_{n,m} \end{pmatrix} \quad (3.36)$$

It is quite clear that each row of matrix in Equation (3.36) adds up to 1, according to Equation (3.23), which means each cell of the matrix is less than or equal to 1. Since each cell represents the probability of a track associated with a hit and Munkres algorithm tries to find the least cost for given matrix, it will return the least likely association between tracks and hits in its current state. In order to prevent this situation, the inverse of the each cell of the matrix in

---

<sup>5</sup> $\beta$ ,  $\beta_0$ , tracks and hits represents the same properties of Section 3.4.1

Equation (3.36) is multiplied by 1 which can be seen in Equation (3.37).<sup>6</sup>

$$Matrix_{r,c} = \sum_{r=1}^n \sum_{c=1}^{m+n} \left( \frac{1}{Matrix_{r,c}} \right) \quad (3.37)$$

In Equation (3.37), divisor may be 0, which is unidentified in terms of mathematics. Actually if any cell of the matrix is 0, it means its associated track and hit has 0 probability of association, which needs to be ignored. In order to solve this problem, when a cell has 0 in it, it becomes  $\infty$  in Equation (3.37) which is the input for extended Munkres algorithm. Since the extended Munkres algorithm tries to find minimum cost for association,  $\infty$  cells are ignored automatically.<sup>7</sup>

When the Munkres algorithm, accepting Equation (3.37) as input, returns results, called  $Result_M$ , the  $Result_M$  is just a column vector which portrays each targets associated hits. But since the input is modified as described above, the output need to be converted to appropriate format. The variable  $n$  is subtracted from the result in order to ignore  $\beta_0$  entries of the input as seen in Equation (3.38). From the resulting column, if the results are less than 1 they are assigned as 0, which can be seen in Equation (3.39). Each 0 member represent the tracks that has no associated hits.

$$Result_M^* = Result_M - n \quad (3.38)$$

$$Associations = \sum_{i=1}^n Result_{M(i)}^* = \begin{cases} Result_{M(i)}^* = 0 & \text{if } Result_{M(i)}^* \leq 0 \\ Result_{M(i)}^* = Result_{M(i)}^* & \text{otherwise} \end{cases} \quad (3.39)$$

The current state of the results, after Equation (3.39), depicts the tracks and their associated hits. When the flow of information is traced back, tracks are  $tracks_{z^*}^{t-1}$  and hits are  $tracks_z^t$ . The  $Results_M^*$  shows the association between  $tracks_{z^*}^{t-1}$  and  $tracks_z^t$ , each 0 element of  $Results_M^*$  shows that associated hits have no tracks, which means related  $tracks_z^t$  has no association with  $tracks_{z^*}^{t-1}$ , and that  $tracks_t^*$  is a new track found at time  $t$ . As a result, for all hits that does not have associated tracks accepted as a new  $track_z^t$  by the agent.

---

<sup>6</sup> $Matrix_{r,c}$  of Equation (3.37) is the same matrix of Equation (3.36)

<sup>7</sup>There exists a special for ignoring the  $\infty$  cells of the matrix

The ASJPDA modified such that it returns both  $Results_M^*$  and  $NewTracks_M$  where  $NewTracks_M$  is derived from zero elements of  $Results_M^*$  which means these hits is not associated with any track.

### 3.5 Kalman Filtering

Kalman filtering, introduced by [7], is one of the most common methods for estimation and filtering. It is a linear, time-invariant estimator with known state dynamics where Gaussian noise is introduced for measurements. The KF works in two phases where it iteratively updates its state estimates. At each time step, KF provides better estimates. In general, the first time step in KF provides the most error prone estimates.

In the first step of KF at time  $t$ , it predicts the state of the given system with their uncertainties. Then, in the second step, at time  $t + 1$ , it generates a Kalman gain according to observation and uses Kalman gain as a correction factor for estimation. The predictor of a Kalman filter is described in Equation (3.40), the Kalman gain is computed according to Equation (3.41) and corrector is applied according to Equation (3.42).

$$\begin{aligned}\tilde{x}(k+1) &= A \cdot \hat{x}(k) \\ \tilde{\Sigma}(k+1) &= A \cdot \hat{\Sigma}(k) \cdot A^T + Q\end{aligned}\tag{3.40}$$

$$K = \tilde{\Sigma}(k+1) \cdot C^T \cdot [C \cdot \tilde{\Sigma}(k+1) \cdot C^T + R]^{-1}\tag{3.41}$$

$$\begin{aligned}\hat{x}(k+1) &= \tilde{x}(k+1) + K \cdot [y(k) - C \cdot \tilde{x}(k+1)] \\ \hat{\Sigma}(k+1) &= (I - K \cdot C) \cdot [A \cdot \tilde{\Sigma}(k+1) \cdot A^T + Q]\end{aligned}\tag{3.42}$$

The KF requires the system dynamics to be known, which is applied to the filter, Equation (3.40), via  $A$ . In this paper,  $A$  is used as in Equation (2.4). Also, the  $R$  in Equation (3.41) stands for estimated measurement errors which is initialized as in Equation (3.43) in this paper. The  $R$  in Equation (3.41) uses the  $\Sigma_t$  of the target as its upper left sub matrix with its predefined diagonal matrix of  $Error_{Estimate}$  as its lower right sub matrix. The rest of the  $R$  matrix is 0 as seen in Equation (3.43). It should be noted that upper left corner of  $R$  matrix changes at each time step according to *covariance* of the track.

$$R_{6,6} = \begin{bmatrix} \Sigma_t & 0_{[3,3]} \\ 0_{[3,3]} & \text{diag}(\text{Error}_{\text{Estimate}}) \end{bmatrix} = \begin{bmatrix} \Sigma_{t_{x,x}} & \Sigma_{t_{x,y}} & \Sigma_{t_{x,z}} & 0 & 0 & 0 \\ \Sigma_{t_{y,x}} & \Sigma_{t_{y,y}} & \Sigma_{t_{y,z}} & 0 & 0 & 0 \\ \Sigma_{t_{z,x}} & \Sigma_{t_{z,y}} & \Sigma_{t_{z,z}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \text{Error}_{V_x,V_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & \text{Error}_{V_y,V_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & \text{Error}_{V_z,V_z} \end{bmatrix} \quad (3.43)$$

The matrix  $Q$  in Equation (3.42) is called dynamic noise matrix. In this paper, it is initialized as if in the Equation (3.44) where the  $\text{Noise}_{\text{Error}}$  of Equation (3.44) is predetermined in the initialization phase of KF.

$$Q_{6,6} = \text{diag}(\text{Noise}_{\text{Error}}) = \begin{bmatrix} \text{Noise}_{\text{Error}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \text{Noise}_{\text{Error}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \text{Noise}_{\text{Error}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \text{Noise}_{\text{Error}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \text{Noise}_{\text{Error}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \text{Noise}_{\text{Error}} \end{bmatrix} \quad (3.44)$$

In this paper, KF is used for each track after JPDA is initialized and  $\text{Results}_M$  and  $\text{NewTracks}_M$  of Section 3.4.2 are acquired. The KF is used for each track in order to further improve their state estimates at each time step. When the flow of information is traced back according to Section 3.1, it should be noted that the  $\hat{t}$  and the  $\Sigma_t$  of the track itself is used as an input at each time step. The  $\Sigma_t$  of tracks is used in Equation (3.43) while the  $\hat{t}$  of the track is  $y(t)$  of Equation (3.41).

KF uses previous steps'  $\hat{x}(t-1)$  and  $\hat{\Sigma}(t-1)$  as input parameters in time step  $t$ . Also, KF estimates the tracks'  $\hat{t}$  and  $\Sigma_t$  at each time step via Equation (3.42). tracks  $\hat{t}$  and  $\Sigma_t$  are updated according to Equation (3.42).

Since KF requires the previous step's  $\hat{x}(t-1)$  and  $\hat{\Sigma}(t-1)$ , it is initialized for per track and the  $\hat{x}(t-1)$  and  $\hat{\Sigma}(t-1)$  are carried over according to the result of Section 3.4.2. The associated

tracks at time step  $t - 1$  and  $t$ , are used for carrying over the KF parameters of associated track at time  $t - 1$  to track at time  $t$ .<sup>8</sup> As a result, KF can preserve its parameters at each time step which is crucial for KF.

---

<sup>8</sup>The tracks at time  $t - 1$  are  $tracks_z^{t-1}$  of Section 3.4.1 and the tracks at time  $t$  are  $tracks_z^t$  of Section 3.4.1

THIS PAGE INTENTIONALLY LEFT BLANK

---

## CHAPTER 4:

### Simulation Results

---

The goal of the simulation is to evaluate the strategy introduced for performing MTT in different scenarios and to identify performance issues. Each scenario introduced different challenges for MTT.

#### 4.1 General Outline

We designed the simulation architecture presented in this thesis to address the variety of multi-target tracking applications and configurations. The main goal of the simulation is to provide realistic detections to the simulated agents while recording their respective track information and comparing these tracks against true target information. The accuracy of the track generation process and accurate identification of tracks in subsequent time steps are the two criteria used as performance metrics. For each situation, different systems parameters, such as target and agent numbers, sensor characteristics, and environment size, are provided as simulation inputs. In addition to their impact on multi-target tracking performance, these parameters affect the overall simulation in terms of computational complexity and time.

For post analysis, the simulation maintains a log tracking performance of each individual agent over the evolution of the scenario. One of the key performance metric is the pairwise Euclidean distance between each generated track and each of the targets' true state (i.e., position and linear speeds) in the environment. The performance is evaluated for the minimal-distance pair, that is, for the target location closest to the given track location, capturing the discrepancy between the agents' common operational picture and the true state of all targets. It should be noted that although the simulation maintains a mapping of detections to tracks (e.g., for evaluation of the clustering algorithms), it does not record the association of the noisy detections to the targets from which they are generated. This focuses the study presented in this thesis on the role of the algorithmic framework for creating and maintaining high level information, i.e., tracks, rather than managing and processing lower level information such as detections.

We rigorously study and present results for four scenarios to test the simulation and the performance of the integrated algorithms presented in Chapter 3. Each scenario progresses in complexity, as assumptions are relaxed and additional considerations are incorporated, showing their impact on performance metrics.

Section 4.2 presents a baseline scenario under simplifying assumptions of deterministic initial conditions for both agents and targets and no uncertainty in their transit nor in detections generated. Section 4.3 preserves only the assumption of perfect detections (that is, now all agent and target initial conditions are randomized and move according to a noisy constant velocity model), and also increases the number of agents and targets to examine the implementation’s scalability. In Section 4.4 and Section 4.5, we introduce false positive and false negative detections to the simulation, in addition to other sources of noise (e.g., initial conditions, motion of both agents and targets), and evaluate their impact on the multi-target tracking performance. All other parameters for simulation are held fixed across all four scenarios, such that the impact of certain variables (e.g., randomization, uncertainty, scale) can be observed in each of the investigated scenarios.

The predefined parameters shared by each simulation are described in Table 4.1. All of these parameters are kept same for each simulations in order to observe the affects of varying situations impact on mission goals. The parameters not represented in Table 4.1 are assigned arbitrarily for each simulation and described in each simulation’s own section.

All parameters in Table 4.1 is acquired for each scenario.

## 4.2 Scenario One – Baseline

### 4.2.1 Scenario One Description

The first scenario examines a baseline case to demonstrate the characteristics of the component algorithms developed in this thesis. This scenario contains three agents and three targets, and the simulation time is assumed to be 50 turns. In this simplified initial study, the motion models of both allies and targets are assumed known and perfect (i.e., , with  $\omega$  as 0 and  $\rho$  as 0 for  $\zeta[\omega, \rho]$  of Equation (2.3)). As a result, each agent moves in a deterministic manner.

Each agent state is initialized according to the parameters in Table 4.2, representing deterministic initial conditions and fixed speeds in only one dimension. In order to prevent agents from reaching the boundaries of the simulation and bouncing back, as described in Section 2.3, the simulated boundaries of the environment have dimensions of 250, 100, and 100 units of length for  $x$ ,  $y$ , and  $z$  dimensions, respectively.

As a final simplification, no false positive or false negative detections are injected into the simulation.

<b>Variable</b>	<b>Description</b>	<b>Value</b>
<i>AAXCoor</i>	The bias in x coordinates when initializing ‘Agents’	0.1
<i>EAXCoor</i>	The bias in x coordinates when initializing ‘Targets’	0.9
$V_x$	The bias in $V_x$ coordinates in initialization	3
$V_y$	The bias in $V_y$ coordinates in initialization	0.85
$V_z$	The bias in $V_z$ coordinates in initialization	0.85
$Length_x$	The default length in x coordinates for occlusion	0.2
$Length_y$	The default length in y coordinates for occlusion	5
$Length_z$	The default length in z coordinates for occlusion	0.3
$\mu$	The horizontal FOV of agents	1.0470
$\lambda$	The vertical FOV of agents	0.7853
$r$	The depth of the FOV of agents	70
$X$	The default $\zeta(\mu, \lambda)$ Gaussian distribution for x dimension	$\mu=0, \lambda=1.5$
$Y$	The default $\zeta(\mu, \lambda)$ Gaussian distribution for y dimension	$\mu=0, \lambda=0.7$
$Z$	The default $\zeta(\mu, \lambda)$ Gaussian distribution for z dimension	$\mu=0, \lambda=0.5$
$\rho_x$	The Gaussian distribution for agent states dynamics	0.4
$\rho_y$	The Gaussian distribution for agent states dynamics	0.4
$\rho_z$	The Gaussian distribution for agent states dynamics	0.4
$\rho_{V_x}$	The Gaussian distribution for agent states dynamics	0.3
$\rho_{V_y}$	The Gaussian distribution for agent states dynamics	0.3
$\rho_{V_z}$	The Gaussian distribution for agent states dynamics	0.3
$Q_{x,x}$	The Q matrix of KF	0.15
$Q_{y,y}$	The Q matrix of KF	0.15
$Q_{z,z}$	The Q matrix of KF	0.15
$\lambda$	The default parameter for ASJPA	6.30E-62
$P_D$	The default parameter for ASJPA	0.95
<i>Acceptancelimit</i>	The criterion for extension gate in ASJPDA	160

Table 4.1: Simulation Parameters

## 4.2.2 Scenario One Results

### Individual Agent Tracking Performance

After the simulation executing for 50 turns, the following results are acquired. Consider Agent 1 and its representation of the common operational picture. For agent 1’s first track in its array of all tracks, we identify the true target closest to the given track, and plot the evolution of the position estimates over time in  $x$ ,  $y$ , and  $z$  dimensions, respectively, in Figure 4.1, 4.2, and 4.3.

According to Figure 4.1-4.3, it is clear that the agent is able to keep track of the target quite accurately for each dimension. The agent occasionally loses the track in some time intervals, but as long as it has continuous track knowledge, the state-estimate errors are quite low. When

Agent Type	State Parameters					
#	$Loc_x$	$Loc_y$	$Loc_z$	$V_x$	$V_y$	$V_z$
Agent 1	10	8	55	3.1	0	0
Agent 2	12	10	50	3	0	0
Agent 3	8	12	45	3	0	0
Target 1	15	8	55	3.1	0	0
Target 2	21	10	50	3.1	0	0
Target 3	19	12	45	3.2	0	0

Table 4.2: The initial states of three agents and three targets in Scenario One. Note that the motion of these agents is deterministic.

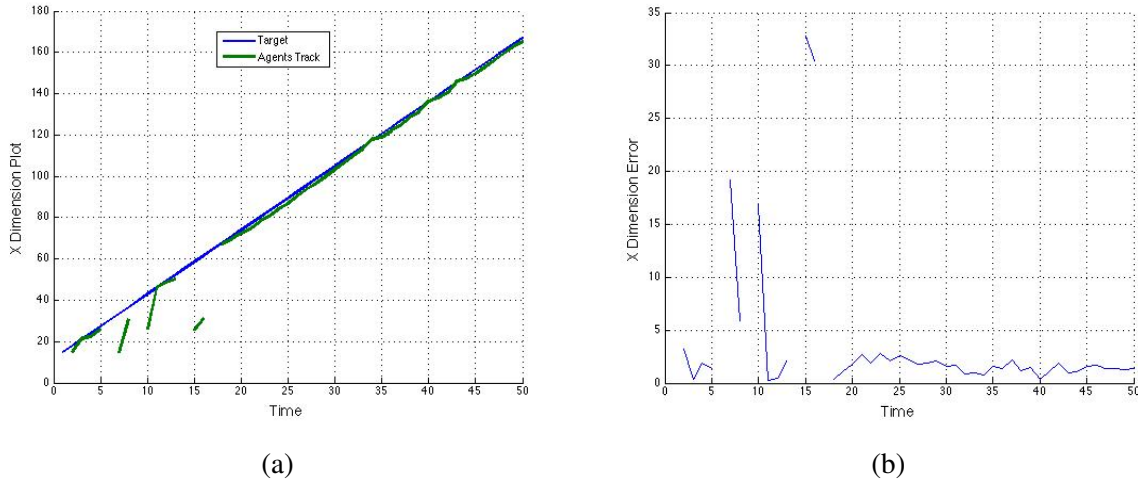


Figure 4.1:  $x$  dimension views of Target 1 and Agent 1's closest track in Scenario One. (a) The  $x$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $x$  dimension error of Agent 1's track and Target 1

the agent loses the target and finds it again after some time, the state estimate error is initially very high but converges quickly within approximately four time steps. As long as an agent keep track of the target, it has very low tracking errors in position, which averages to less than 1.5 meters in each dimension for these continuous track intervals. The agent keeps track of target with average errors over all time (including those where the track is lost and regained) of 7, 0.8, and 1.5 meters for  $x$ ,  $y$ , and  $z$  dimensions, respectively.

Further inspection of the scenario results identifies the element with the greatest impact on target position estimates by the agents to be the Kalman filter for refining track estimates. The KF improves target estimates as long as the prior knowledge of the track is carried over. When a track  $t$  is first generated by the agent, the Kalman filter estimates are initialized with default

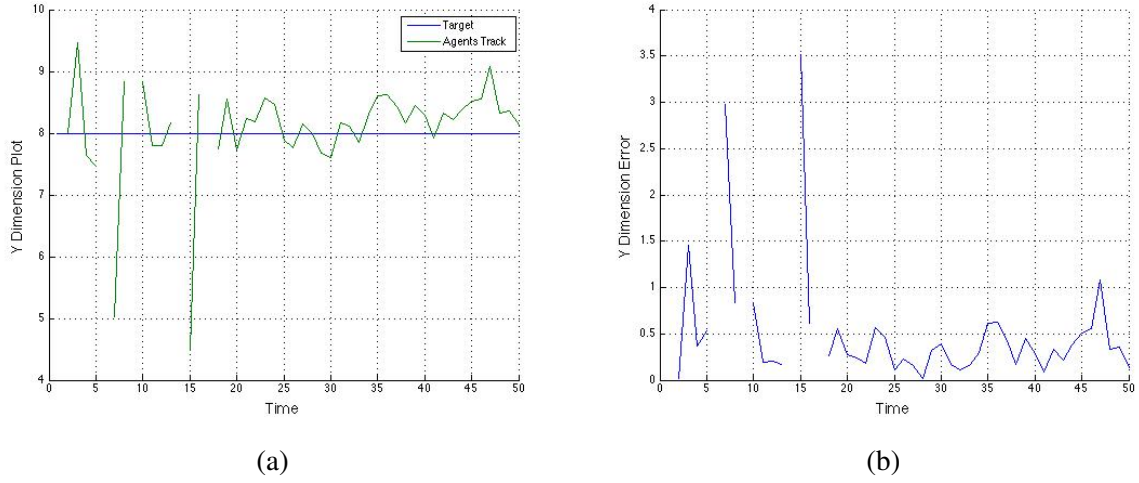


Figure 4.2: y dimension views of Target 1 and Agent 1's closest track in Scenario One. (a) The y dimension plots for both Target 1 and Agent 1's closest track (b) The y dimension error of Agent 1's track and Target 1

parameters representing prior information,  $\hat{t}$  and  $\Sigma_t$ . This leads to initially large errors in the estimate of the track, although the convergence to steady state with better accuracy is rather quick.

### Aggregate Team Tracking Performance

In addition to assessment of tracking performance for individual agents, we inspect the team performance over all the agents. Recall that at each time step, each agent generates tracks with state estimates  $\hat{t} \in \mathbb{R}^6$ . Given the fixed number of targets in the environment, each of which is also represented by their six position- and linear-speed true states, the difference between each track and its associated (i.e., closest) target is computed. Each of these differences at each time step shows how accurately the collective tracks are generated. The average of these differences between tracks and their associated (closest) targets can be seen Figure 4.4, and represents the *aggregate* tracking performance.

Figure 4.4 shows the aggregate error as box plots over all tracks and their associated targets, and though there may be peaks in the statistics on occasional time steps, the average value of the error decreases exponentially as expected after each peak. This is due to the KF improving of tracks' state estimate. Averaging the position errors over the duration of the simulation yields deviations of 7.21, 0.81 and 1.59 meters for  $x$ ,  $y$  and  $z$  dimensions, respectively, whereas the speed errors for  $V_x$ ,  $V_y$ , and  $V_z$  are 2.80, 0.26 and 1.30 meters per second. The greatest errors are

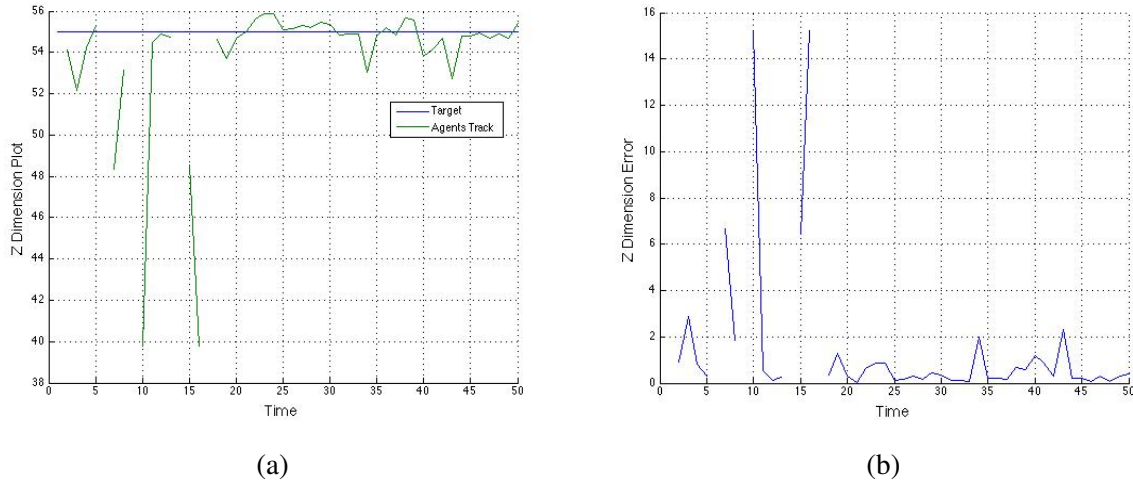


Figure 4.3:  $z$  dimension views of Target 1 and Agent 1's closest track in Scenario One. (a) The  $z$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $z$  dimension error of Agent 1's track and Target 1

found in the  $x$  direction, due to the motion of the targets in this dimension. Note that although there is no noise introduced for agent motions and detections, error can still be introduced by missing detections due to occlusions, improper clustering, errors in track association (e.g., when two targets cross paths), etc. These additional sources of imperfect tracking highlight the challenge of multi-target tracking frameworks, even in restricted cases, as in this scenario.

### Estimation of Number of Targets

Recall that the target count is fixed at three throughout the simulation. However, the agents do not know the true number, but must estimate it based on the observations. As explained in Section 3.4.1, agents attempt to associate tracks at time  $t$  with the tracks at time  $t - 1$ . When one or more tracks appear not to have an associated history, new tracks are initialized, along with a new Kalman filter to refine the track estimates. The average count, over all agents, of newly initialized tracks, labeled “New Initialized Tracks,” and tracks that have associated tracks from previous time steps, called “tracks Associated with History,” are illustrated in Figure 4.5.

Figure 4.5, we see, on average, 2.93 associated tracks and 0.37 new tracks. On average, 3.30 tracks are found throughout the simulation, even if there exists three targets. This shows us that agents generate slightly more tracks than targets, which reflects the results of track-generation methods. Since everyone is sharing all detections knowledge, track generation methods may generate more tracks than really exists. But the number of tracks on average is very close to the

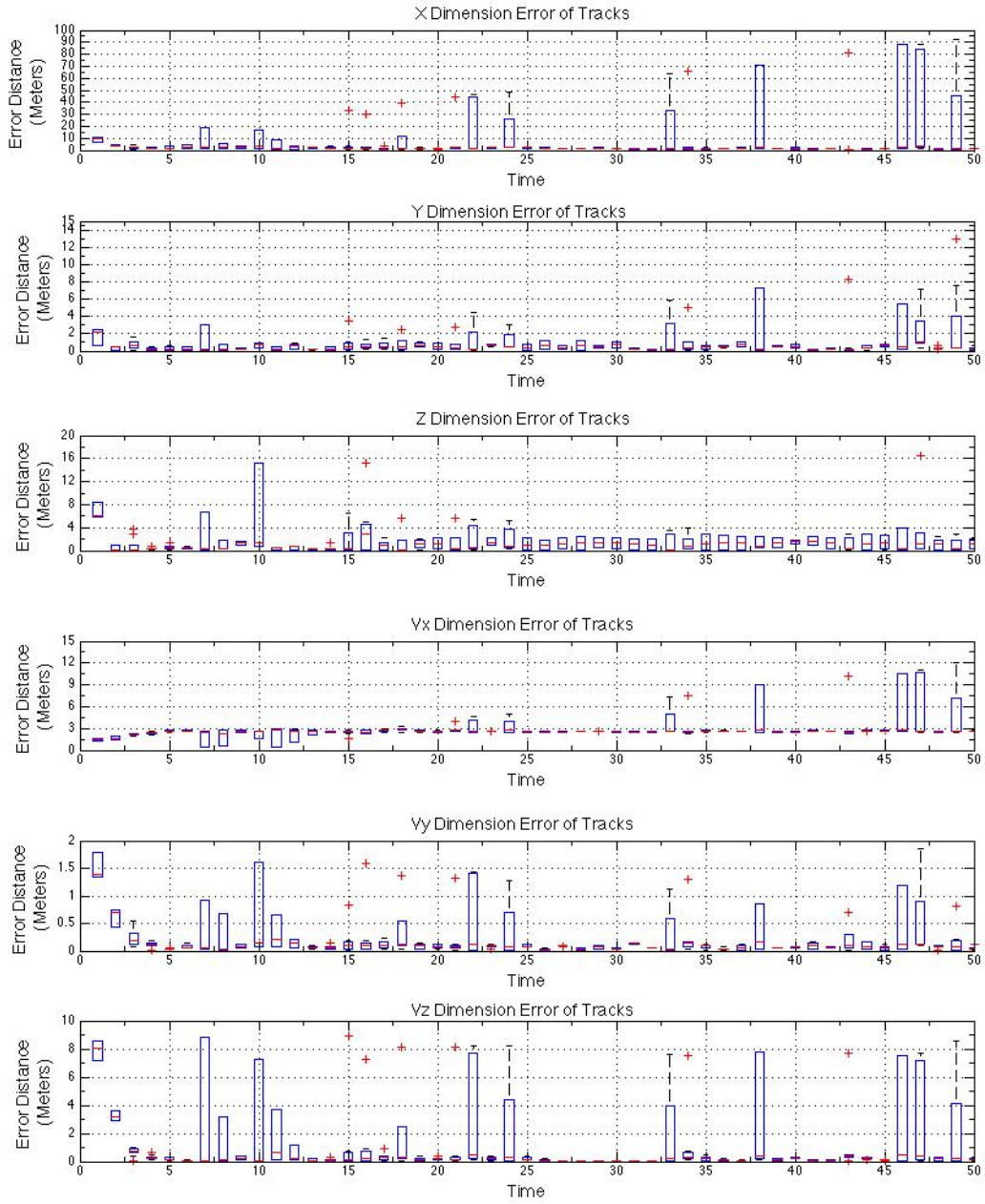


Figure 4.4: Box plots for the tracking error for all state variables of the agents over all tracks and associated target pairings in Scenario One. Average values show exponential convergence to steady state, with large covariances (from newly instantiated tracks) diminishing quickly upon continuous tracking.

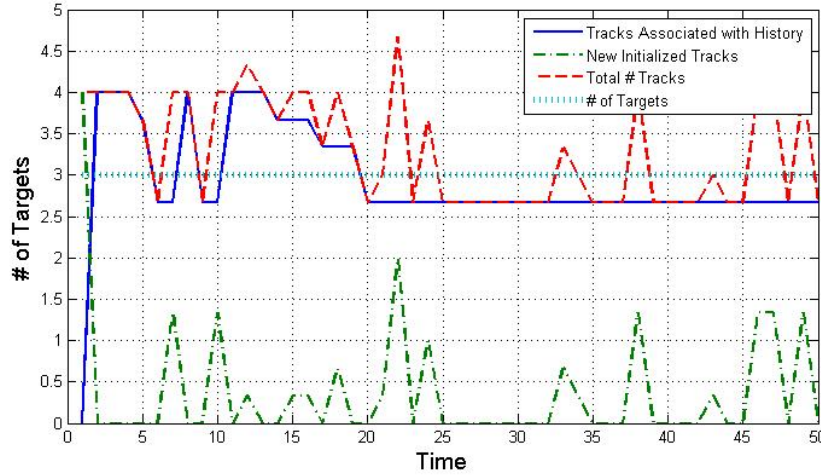


Figure 4.5: The mean number of associated tracks at time  $t$  vs. tracks at time  $t - 1$  for all agents in Scenario One. If tracks have no association, they are accepted as new tracks without prior knowledge.

real number, three. It can be inferred that in this simulation,  $K$ -mean works with great accuracy. Also, the modified L-method, explained in Section 3.3.1.4, generated quite accurate results with the number of detections at most nine.

According to Figure 4.5, at each time step, the average number of tracks not associated with prior track is 0.37. As noted in this scenario, each enemy is in the FOV of at least one agent at all times. The lower the number of ‘New Initialized Tracks’, the more accurate the model is. The best possible outcome for this scenario is that the number of ‘New Initialized Tracks’ is zero (true 70 percent of the time).

Scenario One shows that track estimation is not accurate when targets are first assigned to new tracks. Furthermore, the longer an agent tracks a target, the better the estimate, up to a threshold value for each dimension separately. The model estimates the state of a target quite accurately after it is detected for at least three consecutive time steps. Also, it is observed that the modified L-method provides acceptable results with the low number of detections.

## 4.3 Scenario Two – Many Sensors, Many Targets

### 4.3.1 Scenario Two Description

The second scenario relaxes most of our assumptions from Scenario One by including twelve more agents and targets each (fifteen each in total), adding randomized initial positions and con-

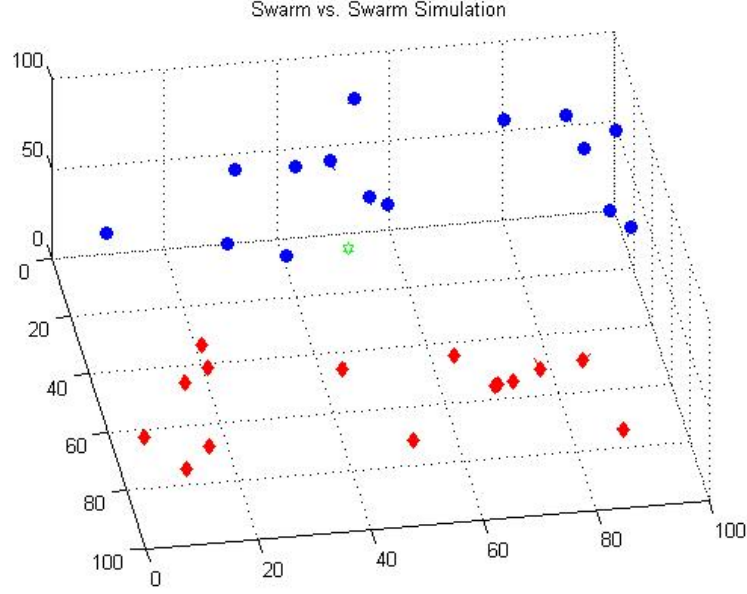


Figure 4.6: An example configuration of simulation at time  $t = 1$  of Scenario Two. Red markers (diamonds) are agents, blue markers (circles) are targets and green marker (hexagram) is a track.

stant speeds, as well as sensor noise. False positive and negative detections are not included. As described in Section 2.2.1, the agents and targets move according to a constant velocity model, perturbed by Gaussian random noise. Such perturbations are more realistic representations of physical settings than Scenario One and account for modeling errors as well as environmental (e.g., wind) effects. With an increased simulation time of 100 turns combined with a smaller bounding box (a cube with width, depth, and height of 100 meters), agents running into the borders reflect off keeping them in the simulation. Figure 4.6 illustrates the initial (randomized) configuration of agents and targets in the presented scenario.

### 4.3.2 Scenario Two Results

#### Individual Agent Tracking Performance

After the simulating for 100 turns, the following results are acquired: To illustrate the performance, consider Agent 1 and its track of the nearest target, denoted Target 1. A graphical representation of the trajectories over time of both Agent 1 and of Target 1 is seen in Figure 4.7. Also depicted are the resulting tracks from all agents associated with Target 1. Visible in Figure 4.7 is the boundary enforcement behavior where the agents reflect off the walls. Despite this nonphysical behavior (to ensure a constant number of agents in the environment), we see the track estimate errors spike as linear trajectory models fail and the Kalman filter must stabilize.

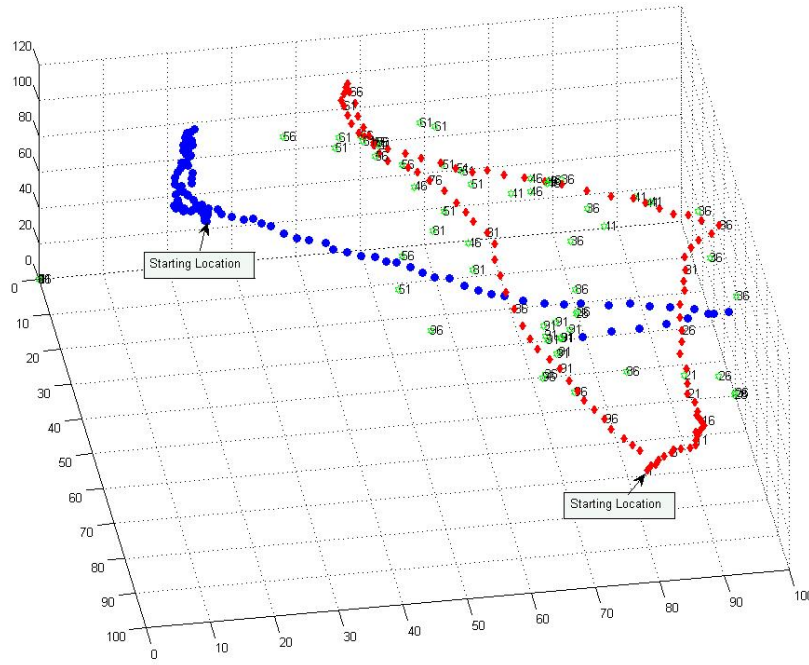
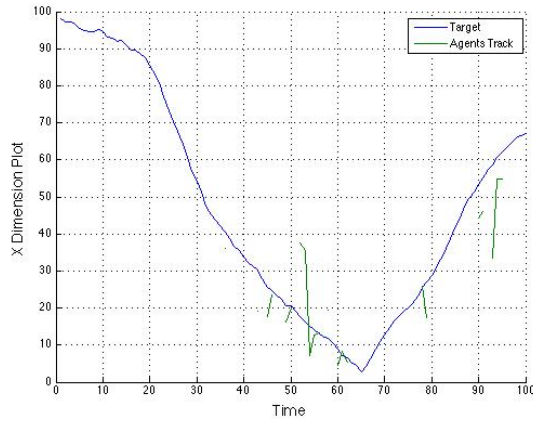


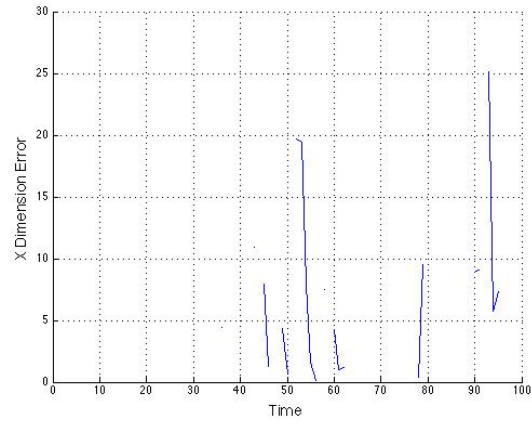
Figure 4.7: The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Two. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of agents and tracks. This figure exemplifies the trace of agents with tracks.

The errors representing the difference in position of the track estimate with the true position of Target 1 is illustrated in Figure 4.8-4.10. Note that Agent 1 does not maintain a consistent track for Target 1 throughout the entire simulation, as illustrated by missing intervals of the track for segments of time. These gaps are caused not only by dropped tracks due to the MTT framework, but also due to the fact that Target 1 may no longer be in the field of view of agents from which to generate (shared) detections. This issue is revisited when trying to estimate the total number of targets in the environment, but merits highlighting the challenge of distributed multi-target tracking from multiple mobile sensors when coverage of the entire environment at every scan (i.e., time step) is not guaranteed (unlike in fixed sensor stations with known coverage footprints).

Similar to the behavior observed in Section 4.2, the agent accurately estimates the target's state after three consecutive time steps of continuous track. As before, convergence to minimal steady-state estimation error is possible for these segments. In this scenario, the agent is seen

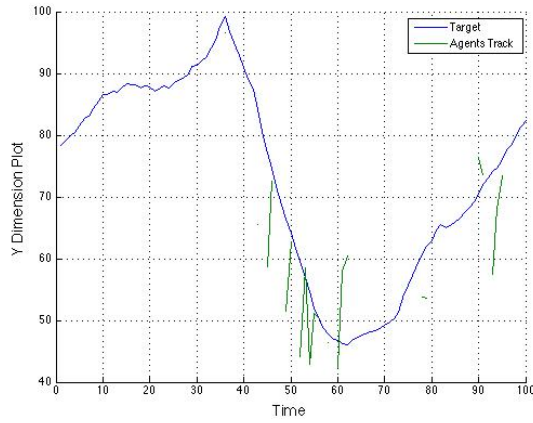


(a)

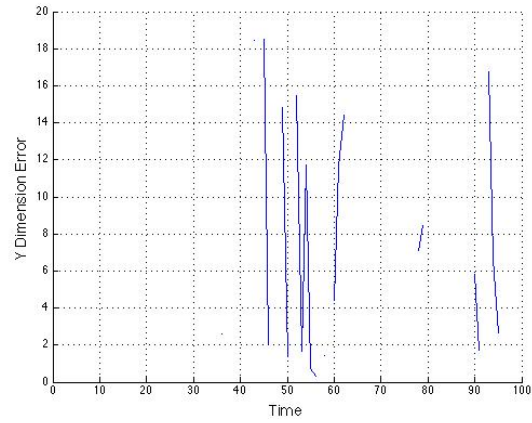


(b)

Figure 4.8:  $x$  dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The  $x$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $x$  dimension error of Agent 1's track and Target 1



(a)



(b)

Figure 4.9:  $y$  dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The  $y$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $y$  dimension error of Agent 1's track and Target 1

to lose track of the target in different time steps, which is expected due to the randomized movements of both agents and targets. For example, the longest duration of continuous track achieved by Agent 1 for Target 1 in this scenario is ten time steps.

For this agent and its track of Target 1, the average position errors over the duration of the simulation run are 7.21, 7.67, and 6.10 meters in  $x$ ,  $y$ , and  $z$ , respectively. The magnitude of the

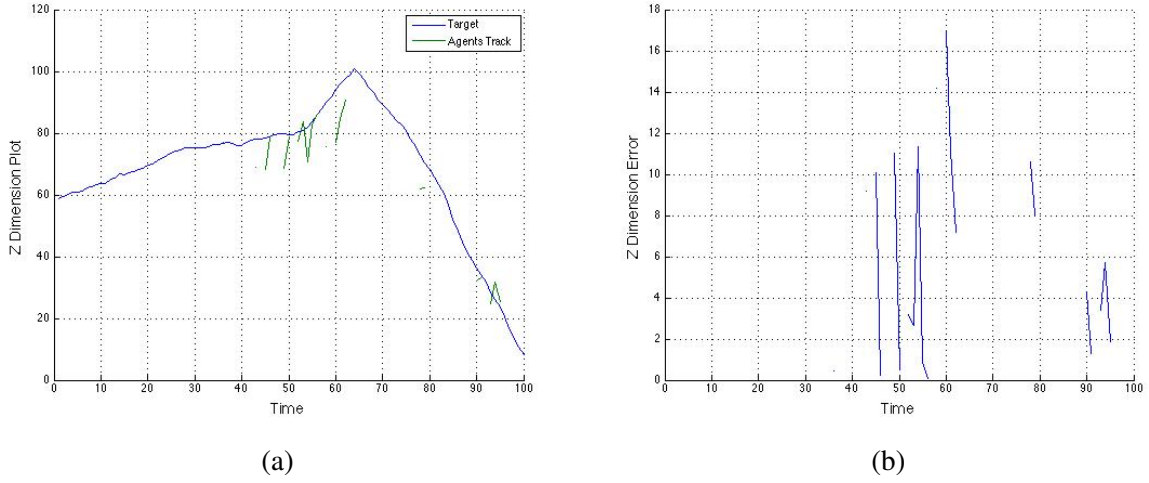


Figure 4.10:  $z$  dimension views of Target 1 and Agent 1's closest track in Scenario Two. (a) The  $z$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $z$  dimension error of Agent 1's track and Target 1

error in the  $x$  estimate is similar to that seen in Section 4.2, though with the addition of random motions in the other coordinate directions, we see an increase in the estimate error as expected.

### Aggregate Team Tracking Performance

We can once again investigate the team performance for overall multi-target tracking error, illustrated in Figure 4.11, which shows the bar plots of the evolution of error for each state variable over time. There aren't bar plots for  $t = 1 \dots 7$ , for as agents have not established tracks.<sup>9</sup> We see that the mean-aggregate estimate error for the state is (10.03, 7.99, 7.17) meters for position and (5.06, 5.63, 5.41) meters per second for linear speeds. As will be seen in Figure 4.12, for any given time  $t$ , if the number of newly initialized tracks exceeds the number of historically associated tracks, the distribution of the errors for each dimension will result in wider bars or variance, due to the reasons explained in Section 4.2 related to the convergence properties of the Kalman filter.

Another key distinction from the previous study is the absence of abrupt peaks in error estimates over the course of the simulation. Given the explicit randomness of motion and its stationarity (i.e., randomness properties doesn't change over time), this scenario offers a more uniform behavior, likely to be more characteristic and comparable to realistic scenarios.

<sup>9</sup>All targets are out of the FOV of the all agents

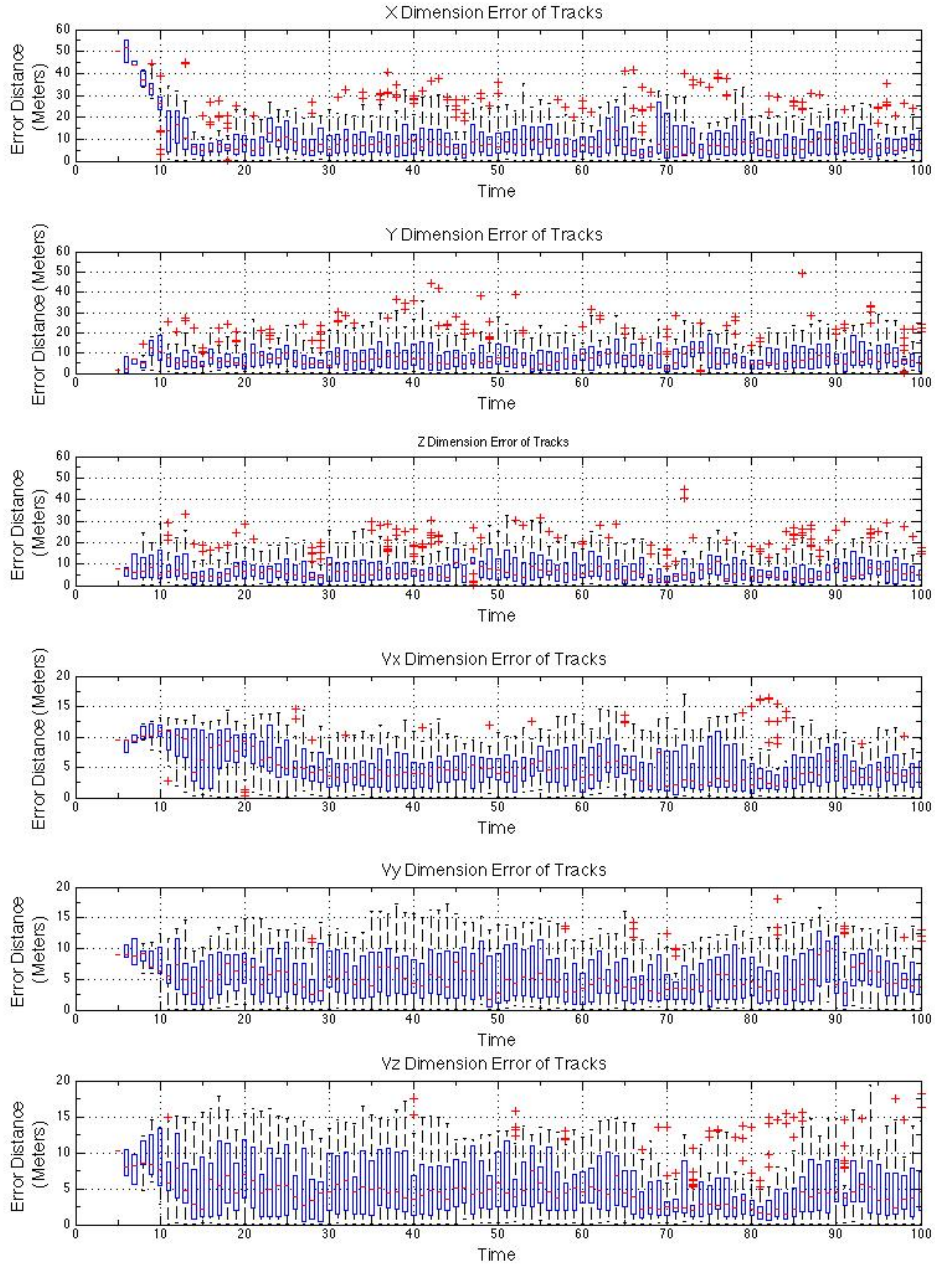


Figure 4.11: Box plots for the tracking error for all state variables of the agents over all tracks and associated target pairings in Scenario Two. Throughout the scenario, there exists an average tracking errors for each dimension that results due to the randomization of all agents.

### Estimation of Number of Targets

The parameters of track association throughout the simulation can be seen in Figure 4.12; on average, there are 4.29 targets tracked by each agent of 15 targets in the scenario. It should be kept in mind that, in all scenarios, every agent shares its detection information with each other.

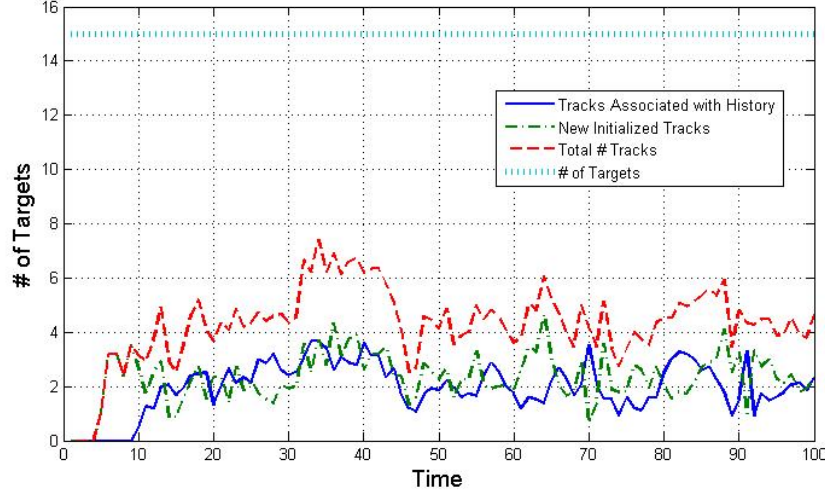


Figure 4.12: The mean number associated tracks at time  $t$  vs. tracks at time  $t - 1$  for all agents in Scenario Two. If tracks have no association, they are accepted as new tracks without prior knowledge

According to Figure 4.12, the average number of tracks associated with history is 1.97 and the average number of newly initialized tracks is 2.32. The ratio between the tracks associated with history and the newly initialized tracks is close to 1 indicating that in each time step, both situations are equally likely. It can be inferred that, even when sharing all detection knowledge, agents cannot continuously keep track of all targets in the simulation (it is observed that targets get out of the FOV of all agents), which causes an increased number of track initialization in each time step.

In this scenario where perfect networking for sharing available detections among the entire team is allowed, we suspected that keeping track of all targets is not achievable without a special area-coverage algorithm implemented among the agents to avoid FOV dropouts, since no explicit control is imparted on the agents to ensure coverage.

Furthermore, though the complexity of this scenario is substantial compared to the previous one, the comparable tracking performance levels indicate that the proposed framework offers good applicability to more realistic and operationally relevant settings.

## 4.4 Scenario Three – Impact of False Detections

### 4.4.1 Scenario Three Description

The third scenario investigates the more challenging context of including both false positive and false negative detections, as formulated in Section 3.2.2. The false positive and false negative detection rates are set to 10%, and 5%, respectively, which implies that there are more spurious detections than true targets unobserved in a given time step of the simulation. All other simulation, environment, sensor, and agent parameters are exactly the same as in Section 4.3, notionally illustrated in Figure 4.7.

### 4.4.2 Scenario Three Results

#### Individual Agent Tracking Performance

As previously, we investigate the ability of a single agent to track a particular target and show the evolution of the track estimate overlaying the true target state, as well as its errors, in Figures 4.13-4.15.

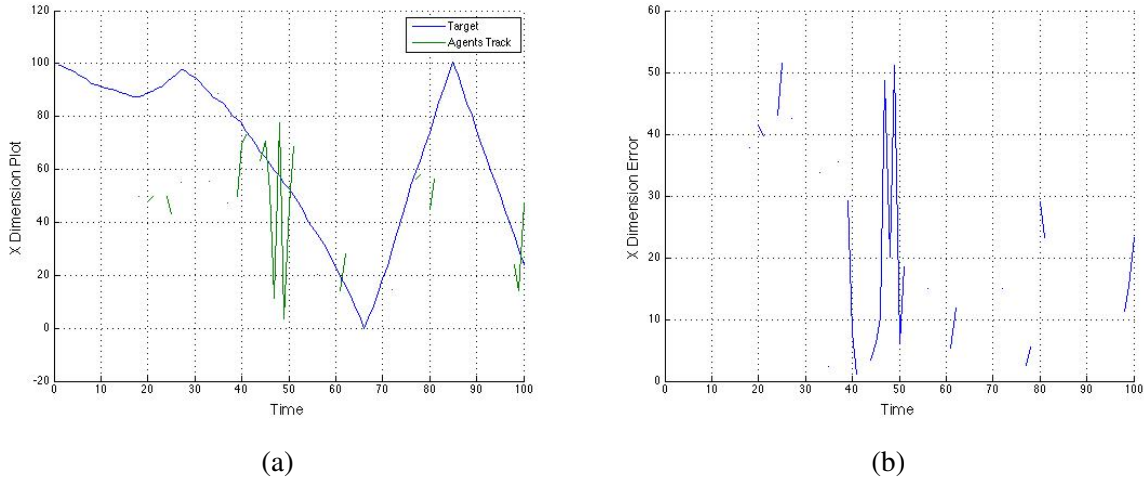
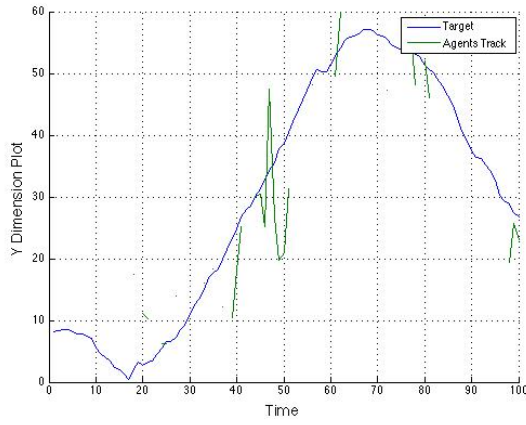
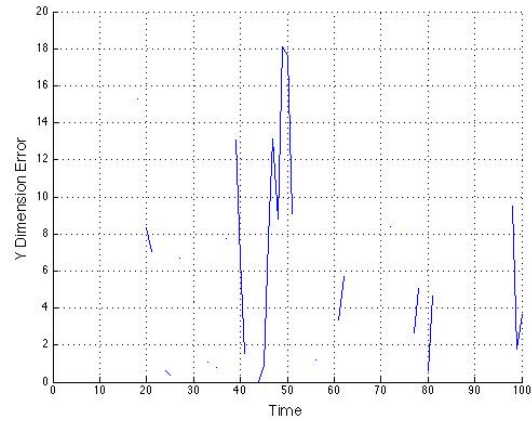


Figure 4.13:  $x$  dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The  $x$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $x$  dimension error of Agent 1's track and Target 1

We find that with the addition of false detections, the estimate errors are noticeably greater than in the previous scenario, with position errors measured 22.61, 6, 11, 7.36 meters in respective coordinate directions. As before, some of these errors are introduced by the occasional lost tracks and/or initialization of new tracks and their impact on the Kalman filtering processes.

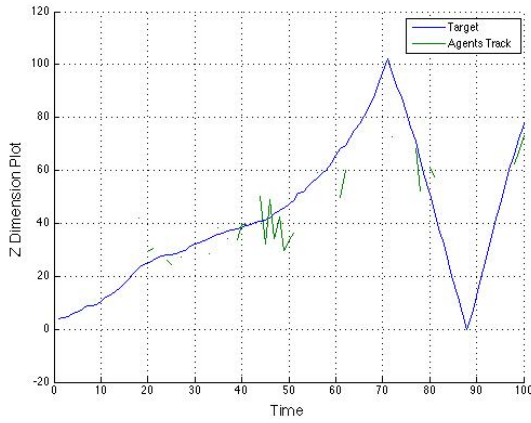


(a)

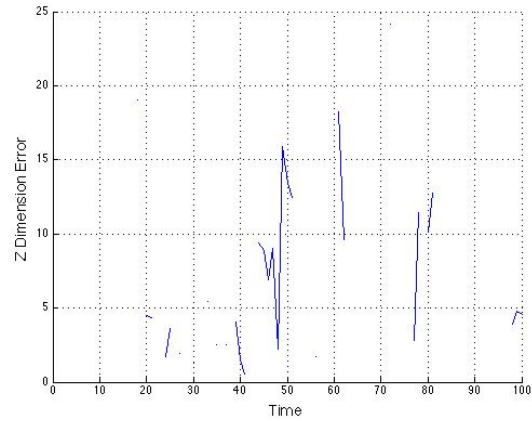


(b)

Figure 4.14: y dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The y dimension plots for both Target 1 and Agent 1's closest track (b) The y dimension error of Agent 1's track and Target 1



(a)



(b)

Figure 4.15: z dimension views of Target 1 and Agent 1's closest track in Scenario Three. (a) The z dimension plots for both Target 1 and Agent 1's closest track (b) The z dimension error of Agent 1's track and Target 1

We would expect the estimates and their errors to decrease and converge over continuous observations of the target. But this is not the case here. In some cases, the error decreases, while in other time intervals, we observe an increase in error, though it appears the track is maintained continuously. Unlike before, the KF no longer seems to converge within approximately three time steps. We conclude that the injection of false detections, and in particular, the high rate of false positive detections, negatively impact this convergence, since additional outlying detec-

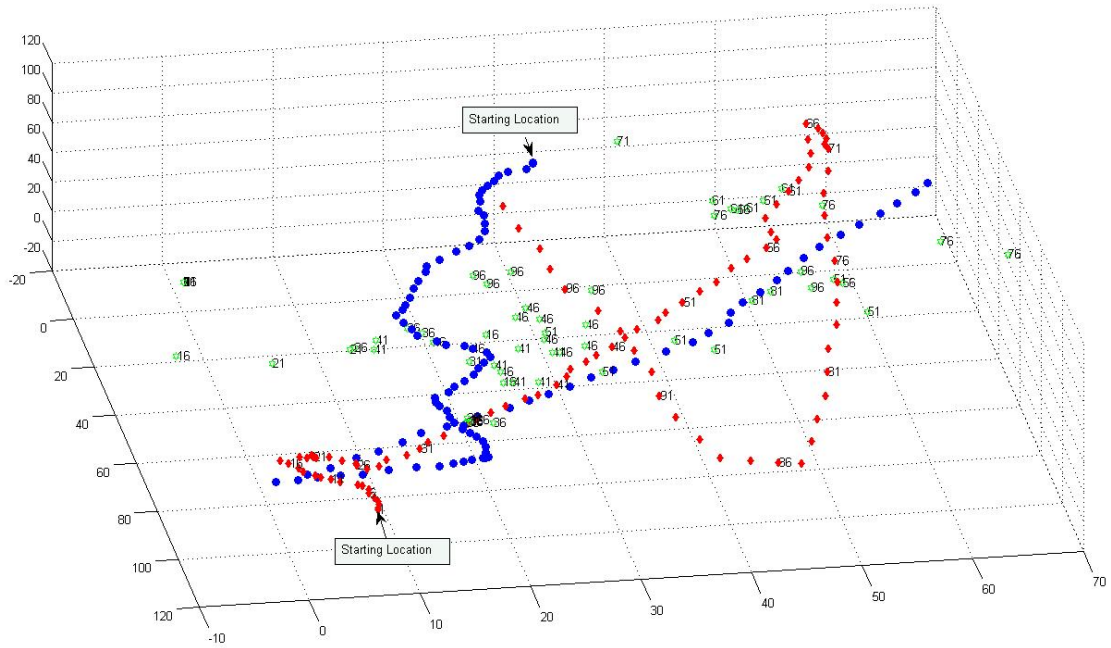


Figure 4.16: The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Three. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of *agents* and tracks. This figure exemplifies the trace of agents with tracks

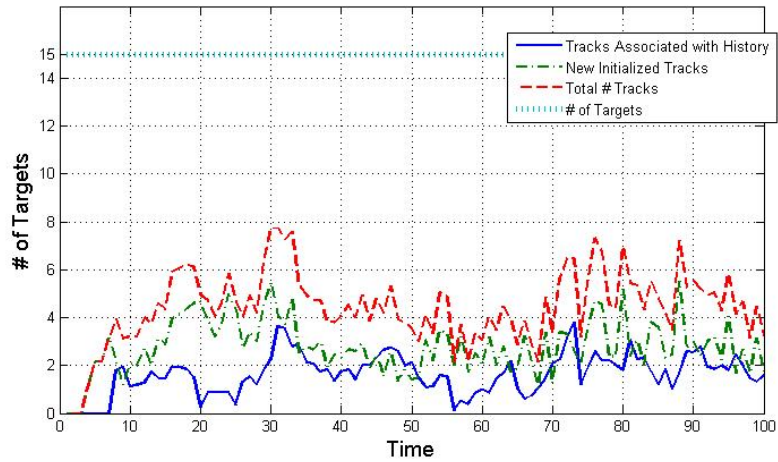


Figure 4.17: The mean number associated tracks at time  $t$  vs. tracks at time  $t - 1$  for all agents in Scenario Three. If tracks have no association, they are accepted as new tracks without prior knowledge

tions not associated with true targets will lead to initialization of more (false) tracks, each with large initial errors and covariances.

### **Aggregate Team Tracking Performance**

Figure 4.18 once again shows the evolution of the track estimate errors for each dimension averaged over all tracks. In this simulation, we expected to have slightly higher outliers in Figure 4.18, due to false positive detections, but we observed that the number of outliers is similar to that in the Figure 4.11. It can be concluded that false detection rate of 10% does not greatly alter the mean-aggregate estimate errors. The amount of false detections is something agents can handle without degraded performance (though there is some performance decline). The false detections are handled by the K-means algorithm without generating too many false tracks (tracks either consists of false positive detections or alterations from the original state of the target due to false negative detections).

The dimensions  $x$ ,  $y$ ,  $z$ ,  $V_x$ ,  $V_y$  and  $V_z$  have 12.20, 9.94, 9.22, 5.49, 6.12 and 5.88 meters of the mean aggregate estimate error respectively. These parameters are slightly higher than in the Figure 4.11.

According to Figure 4.17, on average, it can be seen that there are more newly initialized tracks, as 2.78, than tracks associated with history, as 1.62. On average, each agent has 4.40 tracks, while there are fifteen targets. These results are slightly better than in Figure 4.12. We observed that false detections cause high error results compared to Section 4.3. But, in terms of number of tracks, the agents handled false detections perfectly and generated a more accurate number of tracks, which was unexpected.

According to these results, we inferred that the performance in JPDA algorithm is slightly worse (with a higher number of newly initialized tracks), but the performance of L-method is slightly better (with a more accurate number of tracks) compared to Section 4.3.

The associated track of Agent 1 and Target 1 can be seen in Figure 4.16. This figure depicts the trace of an agent and a target throughout the simulation. Also, it shows the all tracks of all agents associated with the given target, plotted with time intervals.

In Figure 4.16, it can be seen that the target executes high angle rotations at times 16 and 86. There is no track association at time 86, but when the tracks at time 16 are investigated, there is a higher number of aggregate estimate errors, in terms of track location vs. target location, due to the knowledge of the state of the agent.

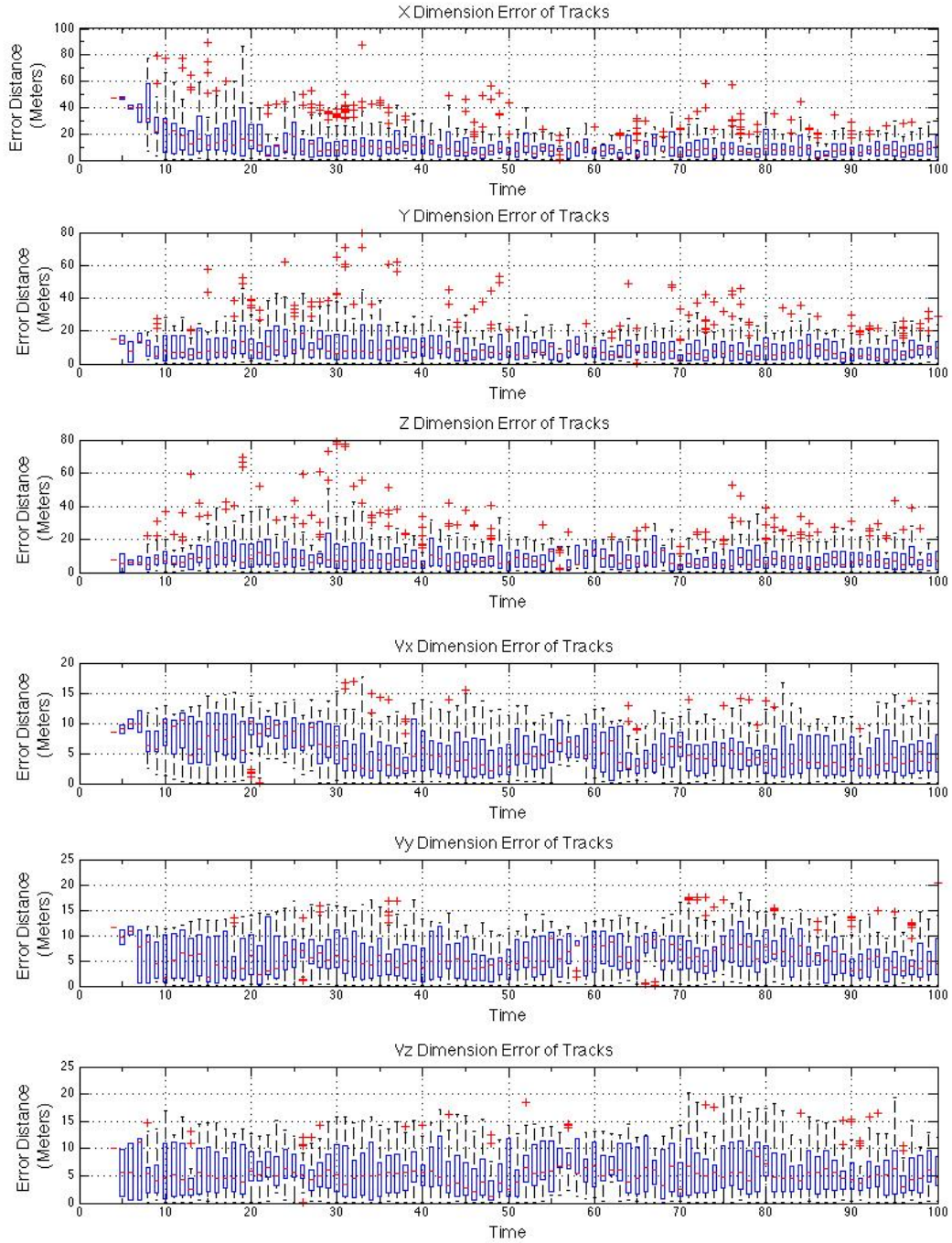


Figure 4.18: This figure, in Scenario Three, depicts the mean error of all agents for each dimension in terms of the difference of each track and its associated target.

In general, Agents assume that the Targets proceed according to its trajectory. A high angle rotation, which is within the limits of the targets, is an unlikely situation that causes deviation from the expected state of the targets. When the time steps between  $t = 16 \dots 35$  are investigated, it has been seen that the agent couldn't manage to associate tracks in given time steps, instead, generate a new track in each 2 or 3 time steps which causes higher mean aggregate estimate errors due to newly initialized KF. The agent manages to keep track of the target after time step 37.

Scenario Three shows that false detections have a negative impact on the mean estimate error of the agents. On average, the  $x$ ,  $y$  and  $z$  dimensions mean-estimate errors that are 2.05 meters higher mean estimate errors while the  $V_x$ ,  $V_y$  and  $V_z$  dimensions have mean-estimate errors that are 0.46 meters higher.

## **4.5 Scenario Four – Sensitivity to False Detections**

### **4.5.1 Scenario Four Description**

The fourth simulation is established in order to provide insight for either false-negative detections or false-positive detections, which have a higher impact on mean-estimate errors. In this scenario, there exists fifteen agents and fifteen targets, and the simulation runs for 100 turns. This simulation has the same setup as Section 4.4 except false-detections ratios.

In this scenario, the linear model of agents, described in Section 2.2.2, is applied and the initialization of the agents is randomized as in Section 4.3. The false positive ratio is set to 5% and the false negative ratio is set to 10%. The false detections are generated as explained in Section 3.2.2.

The simulation border are set to 100 for the  $x$ ,  $y$  and  $z$  dimensions. It is expected that the agents will bounce back from the borders.

### **4.5.2 Scenario Four Results**

#### **Individual Agent Tracking Performance**

After the simulation executes for 100 turns as done previously, the following results are acquired.

The Figure 4.19 throughout Figure 4.21 represents the same information provided in Section 4.3. It should be kept in mind that these figures depict only one agent and one target.

When these figures are investigated, it can be inferred that the mean estimate error for  $x$ ,  $y$  and  $z$  dimensions are 5.61, 9.71 and 11.77 meters respectively. These mean estimate errors are lower than the results in Figure 4.13 throughout Figure 4.15.

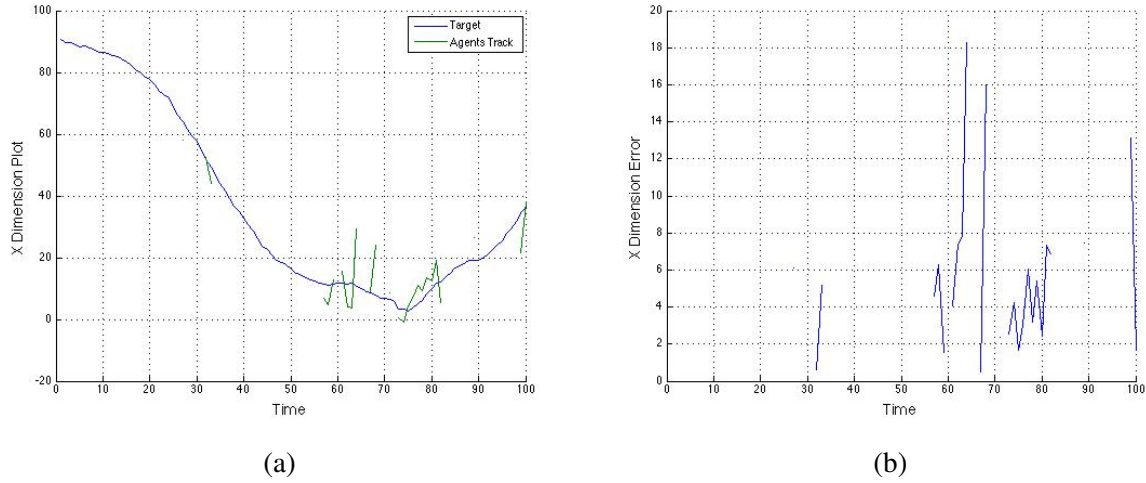


Figure 4.19:  $x$  dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The  $x$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $x$  dimension error of Agent 1's track and Target 1

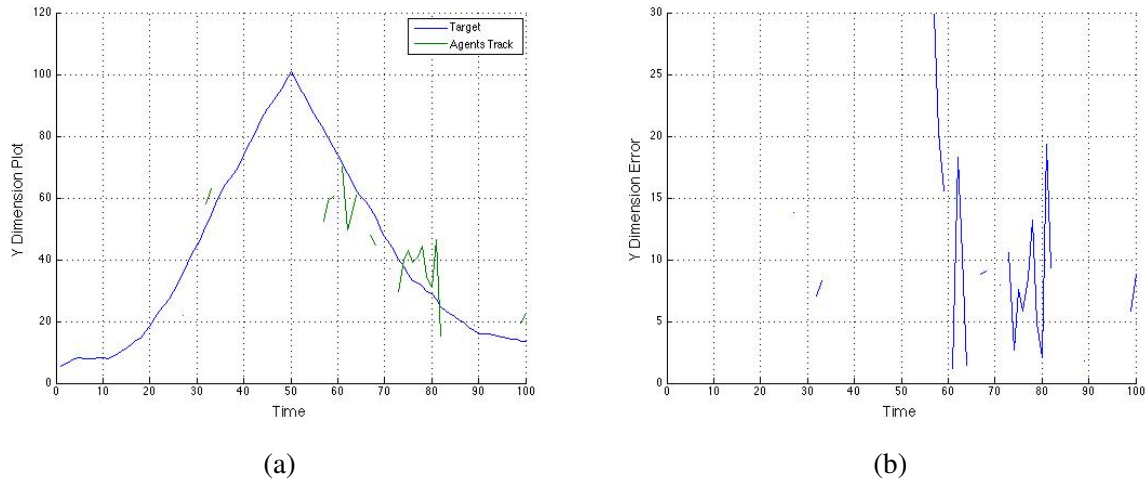


Figure 4.20:  $y$  dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The  $y$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $y$  dimension error of Agent 1's track and Target 1

It is expected that due to the KF, the the mean estimate error will decrease in each concurrent time step. As in Section 4.4, this behavior is not observed for this section. The mean errors

do not decrease with each concurrent time steps; instead it oscillate within a specific value for each dimension. We observed that, in both Section 4.4 and Section 4.5, false detections have a negative impact on KF process of target tracking. False positive detections have slightly higher impacts on KF estimations than false negative detections.

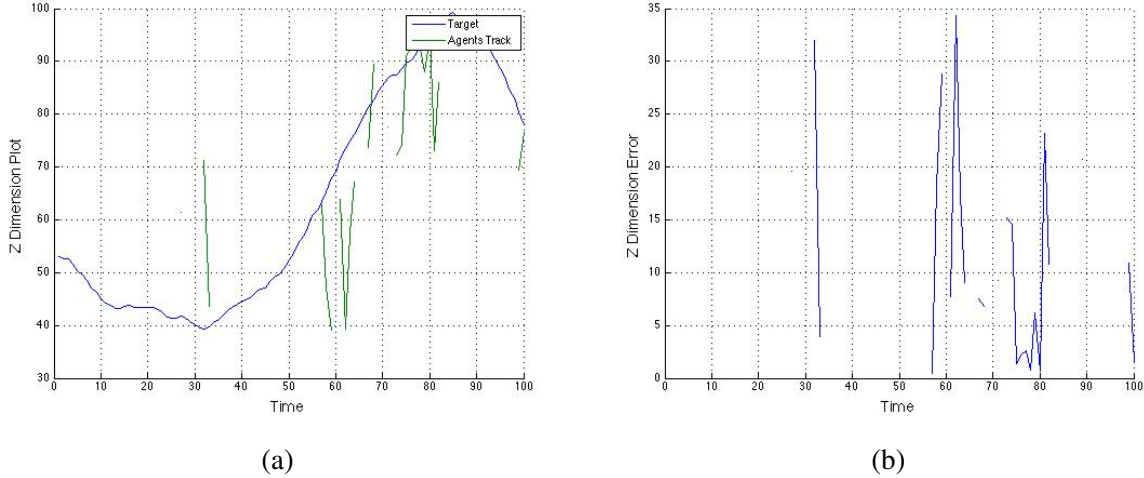


Figure 4.21:  $z$  dimension views of Target 1 and Agent 1's closest track in Scenario Four. (a) The  $z$  dimension plots for both Target 1 and Agent 1's closest track (b) The  $z$  dimension error of Agent 1's track and Target 1

### Aggregate Team Tracking Performance

From Figure 4.24, it can be inferred that the the mean-aggregate estimate error for each dimension oscillates around a specific value, which is similar to that in Section 4.3 and Section 4.4. This simulation also has a similar number of outliers compared with two simulations in Figure 4.24. It can be concluded that false detections do not greatly alter the distribution of the mean-estimate errors. Agents can handle different ratios of false detections without too much trouble, as in Section 4.4.

The dimensions  $x$ ,  $y$ ,  $z$ ,  $V_x$ ,  $V_y$  and  $V_z$  have 11.25, 8.65, 8.40, 5.63, 5.64 and 5.83 meters of the mean estimate error, respectively. These parameters are slightly lower than in Figure 4.18, which means false-negative detections have less impact than false-positive detections.

According to Figure 4.22, on average, it can be seen that there exists more newly initialized tracks, 2.40, than tracks associated with history, 1.13, similar to Section 4.4. Also, in this simulation, on average, the number of targets tracked is 3.53. These parameters show that false-negative detections hamper the performance of track generation methods. The Section 4.4, on

average, has a higher number of targets tracking.

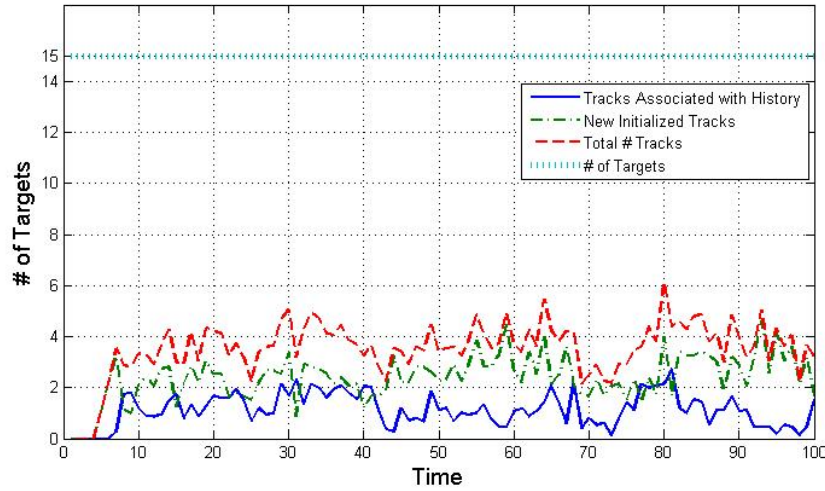


Figure 4.22: The mean number associated tracks at time  $t$  vs. tracks at time  $t - 1$  for all agents in Scenario Four. If tracks have no association, they are accepted as new tracks without prior knowledge

The track of one agent and one target can be seen in Figure 4.23. This figure depicts the trace of two agents throughout the simulation, which are very similar to the plot shown in Figure 4.16.

Simulation 4 shows that, when compared against false positive detections, false negative detections greatly reduces the performance of the JPDA method, but the mean of errors is lower than in Section 4.4. Also, in terms of average number of targets tracked, Section 4.5 has the lowest ration amongst Section 4.3 to Section 4.5

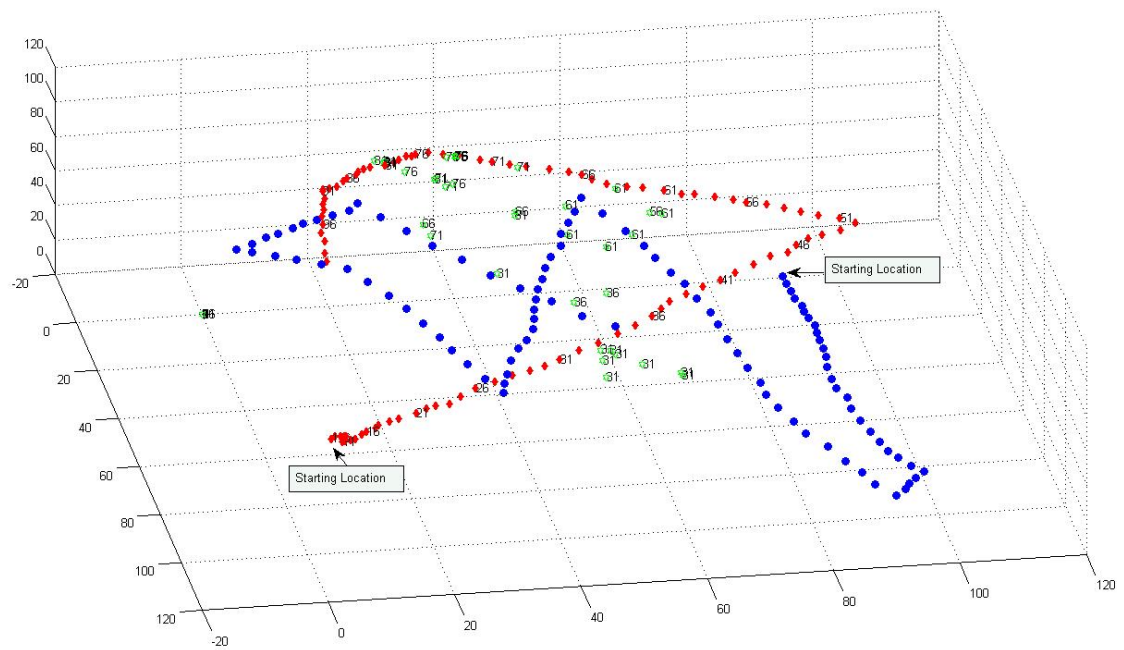


Figure 4.23: The plot of 1 Target (red/diamond) and 1 Agent (blue/circle) throughout Scenario Four. In each five time steps, all tracks of all Agents are plotted (green/hexagram). The markers in this figures are time stamps of agents and tracks. This figure exemplifies the trace of agents with tracks

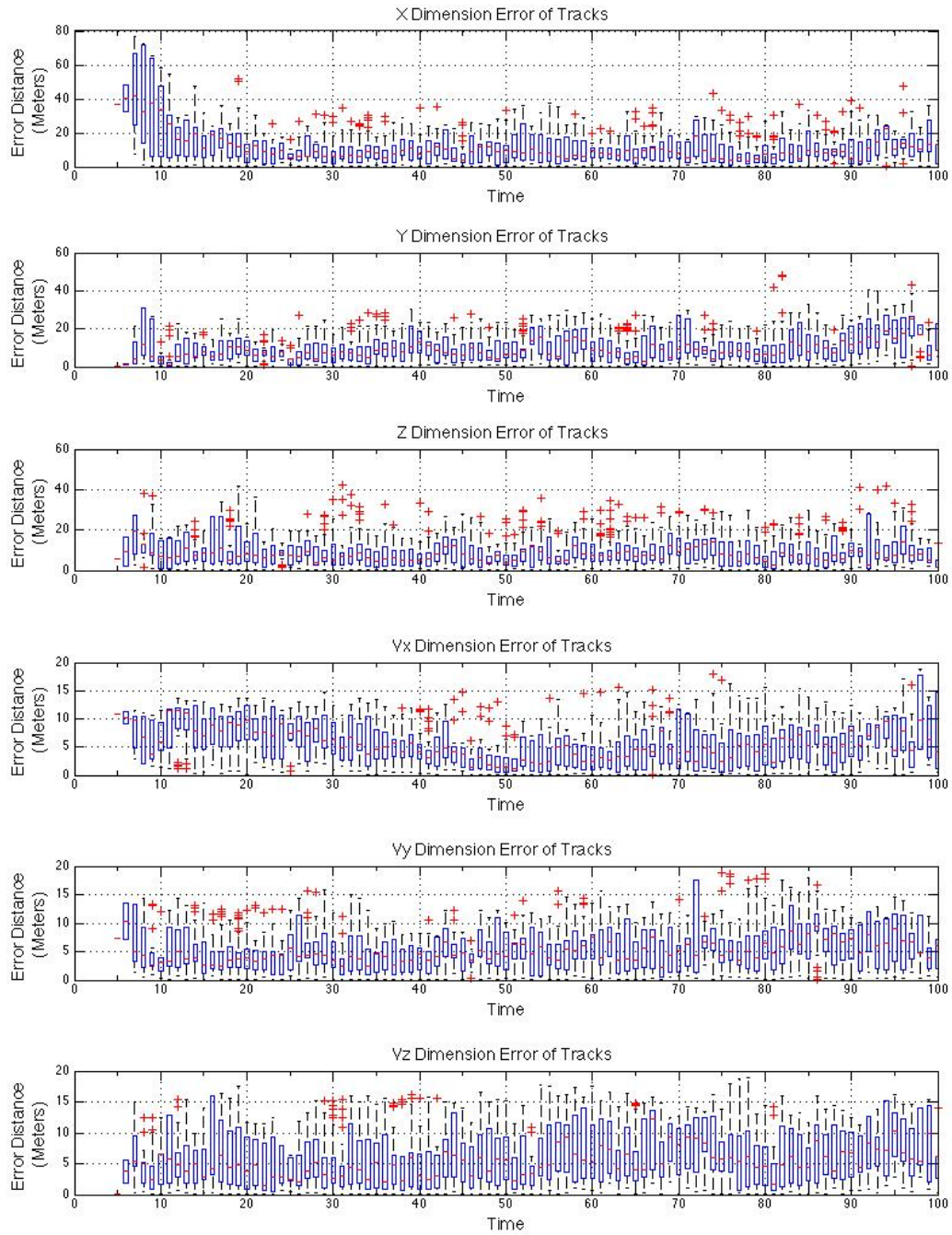


Figure 4.24: This figure, in Scenario Four, depicts the mean error of all agents for each dimension in terms of the difference of each track and its associated target.

THIS PAGE INTENTIONALLY LEFT BLANK

---

## CHAPTER 5:

### Conclusion and Future Work

---

#### 5.1 Conclusion

The main objective of this thesis is to present a novel method for generating global common operational picture for swarm-versus-swarm, unmanned aerial systems by applying well-known methods while providing an autonomous approach that works individually for each agent without a centralized communication node requirement and analyzing their performance for the distributed multi-target tracking (MTT) problem from multiple mobile platforms. The MTT framework integrates a sequence of algorithms to collectively provide this picture. Given various approaches, each has its drawbacks in different contexts. This thesis presents one approach that has relevance to the intended mission, with realistic considerations and emphasis on technical rigor.

In the presented framework, each agent acquires its detections (e.g., from simulated onboard sensors) and generates tracks via a K-means clustering method, providing estimates of the targets' states in the environments. Data association between existing tracks is performed using the augmented suboptimal joint-probability data-association algorithm, in conjunction with the extended Munkres algorithm. Subsequently, tracks are refined or newly instantiated with application of a Kalman filter to compute the state error estimates and their covariances.

The simulation software generates two types of agents, namely those that are the trackers (called Allied Agents) and those to be tracked (called Enemy Agents). A sensor model incorporating noisy detections as well as occlusions is presented, as well as a dynamics model of agent motion that is also subject to noise. Given these elements, each agent works with its networked teammates to construct the common operational picture. The simulation records and summarizes relevant data to provide statistical results on the performance of the framework.

The main measure of performance of the team of sensing agents is the targets' state estimation errors, that is, the difference between the perceived state represented by a set of tracks and the true state of targets (provided by simulation). Four different scenarios are constructed for numerical studies, which investigate different factors which may impact the tracking performance and are relevant to real-world applications:

- Scenario One provides a baseline case to examine the performance of the proposed MTT approach for three agents tracking three targets in the absence of uncertainty, in detections and motion in a relatively unconfined environment.
- Scenario Two examines a larger engagement between fifteen agents vs. fifteen targets, including a noisy model for the constant velocity motion dynamics, as well as imperfect detections representing true targets perturbed by Gaussian noise.
- Scenario Three extends the previous investigation with the inclusion of false positive and false negative detections, with error rates of 10% and 5%, respectively.
- Scenario Four further studies the sensitivity of the false detection error rates, now considering reversed biases with a false positive detection rate of 5% and a false negative detection rate of 10%, still considering the fifteen vs. fifteen setting.

Scenario	Average State Estimate Errors					
#	$x$	$y$	$z$	$V_x$	$V_y$	$V_z$
Scenario One	7.21	0.81	1.59	2.80	0.26	1.30
Scenario Two	10.03	7.99	7.17	5.06	5.63	5.41
Scenario Three	12.20	9.94	9.22	5.49	6.12	5.88
Scenario Four	11.25	8.65	8.40	5.63	5.64	5.83

Table 5.1: Summary of the average state-estimate errors for each of six state variables (positions and linear speeds) from simulation studies for four different scenarios, described in Section 4.

The aggregate tracking performance of each scenario is summarized in Table 5.1. Insights from these simulation studies include the need for a good detection-clustering algorithm, which is the leading step in generating target tracks.

In this paper, the first simulation is designed to generate results such that targets are in the FOV of at least one agent for any discrete time. When the simulation one is analyzed, in which agents keep track of targets all the time, the mean error of agents is lowest. It is clear that preserving the

Scenario		Average number of Tracks		
#	# Targets	#	Newly Initialized	Historical
Scenario One	3	3.30	0.37	2.93
Scenario Two	15	4.29	2.32	1.97
Scenario Three	15	4.40	2.78	1.62
Scenario Four	15	3.53	2.40	1.13

Table 5.2: Summary of average number of track computation for each scenario. It should be noted that number of targets is fixed throughout the simulation.

targets in the FOV provides the best track estimations, which is only guaranteed in Section 4.2.

The studies highlight that false detections negatively impact the tracking performance of the multi-sensor network, though false positive detections appear to result in greater tracking error than their false negative detection counterparts. Additionally, the sensitivity of the track data-association algorithm, balancing the use of existing historical tracks versus the instantiation of new tracks, plays a significant role in the average error, since the Kalman filter provides advantages only after it converges after continuous tracking is maintained. Rather than the false detections, the uncertainties due to dynamics noise (e.g., the random movements of the targets) is a greater challenge in data association than false detections, because targets exiting and entering the collective sensor field of view drop or create tracks, degrading the aggregate tracking performance. These insights can be used to improve not only existing capabilities for MTT, but to develop future requirements on such multi-target tracking systems, specifically leveraging a network of mobile agents such as a swarm of unmanned aerial systems.

The number of targets tracked by each agent plays an important factor in determining the accuracy of track generation in each time step. It is observed that in both the absence of false detections (Section 4.3) and high ratio of false-positive detections (Section 4.4), the mean number of targets is very close: 4.29 and 4.40 respectively. However, when a high ratio of false-negative detections is introduced to the simulation (Section 4.5), the ratio drops 3.53. In terms of number of targets tracked, it is quite amazing that agents handle false-positives detections with great accuracy, while suffering from the false-negative detections.

It is observed that the presence of false detections affects the performance of the Kalman filter. When all detections are true detections, the KF generates very accurate estimations only with four consecutive time steps of target tracking. In each time step, the KF reduces the mean of error greatly, up to a certain value, for each dimension just in four consecutive times of detection, as seen in Section 4.2 and Section 4.3. But when false detections are introduced, Section 4.4 and Section 4.5, KF cannot estimate the targets with great accuracy, even if continuous tracking is established. Instead of decreasing the mean of error in each consequent step, the KF provides an oscillating mean of errors for each dimension.

In all four simulations, target estimations are generated by the KF. It is observed that, for any scenario, when a target is initialized, the KF generates the highest mean of error. It is clear that losing track of a target hampers the performance of agents.

The method for selecting the best  $k$  has an crucial impact on generating an accurate number of targets. In this paper, a modified L-method is used for best  $k$  selection for number of detections less than 20. It turns out that the modified L-method provides quite accurate results, as seen in Section 4.2, the mean number of targets.

## 5.2 Future Work

In this scenario where perfect networking for sharing available detections among the entire team is allowed, it is suspected that, without a special area-coverage algorithm implemented among the agents, keeping track of all targets is not achievable with the current framework, since no explicit control is imparted on the agents to ensure coverage. This issue causes higher errors in both individual and aggregate tracking, since the target cannot be guaranteed to remain in the collective field-of-view continuously over the course of the scenario. There are several areas that can be extended in future studies to better address the requirements of performing multi-target tracking where the sensors are mobile themselves. First, refinement of the detection clustering algorithm according to the anticipated or nominal number of detections in practice would likely provide substantial improvement to the tracking performance. In particular, the K-means approach using the L-method to determine the number of clusters provided a computationally cheap and attractive approach, especially knowing that the ultimate goal is to track agents in a swarm of UAVs; however, for scenarios where fewer detections are available (e.g., when there are fewer targets or fewer sensors), this method appears to be less effective. Nonetheless, additional methods beyond those investigated in this thesis may highlight better performing implementations.

Another significant component of the MTT process is the task of associating tracks over time, such as the Augmented Suboptimal Joint Probability Data Association algorithm implemented in this work. Given the dependence on only the previous track (rather than an extended history), the ASJPDA algorithm suffers when a track is lost, that is, new tracks are necessarily initialized in the next time step, which increases the average error due to the initial conditions of the implemented Kalman filter. Instead, future work could investigate the advantages of using additional past information at the expense of computational complexity to mitigate such shortcomings.

Additionally, the  $\lambda$  and  $P_D$  parameters of the ASJPDA have been observed to play an important role on the  $\beta$  and  $\beta_0$  terms, which directly influence how well tracks are associated with previous ones. For the presented work, these naïvely chosen parameters are held constant throughout the duration of the simulation, as well as across the scenarios studied in this thesis. In future work,

perhaps depending on the specific scenarios of interest, the  $\lambda$  and  $P_D$  parameters can be analyzed and appropriately selected to provide better ASJPDA results for track associations.

Further, the information from previous tracks can be used to help prune the detections that influence the clustering algorithm and thus the association algorithm. For example, one could identify detections that are more than some threshold away and no longer included in the calculation of the specific track's estimate mean and covariance, thereby improving the tracking performance. The detections discarded in this manner could remain unassigned, could trigger the creation of a new track, or could possibly be assigned to other tracks.

A future enhancement to the simulation software would be for each agent to also possess knowledge of the true target labels or ID for all detections (which is only known to the simulation) and for associated tracks. As the primary focus of the thesis was on collective tracking performance, rather than on individual unique target identification and tracking of specific targets, the simulation did not rely on a labeling system for targets that could be stored for further analysis. Future versions of this software could incorporate such data to aid in conducting additional simulation analysis studies.

Also, the simulation software does not possess the knowledge of targets that are in the cumulative FOV of the agents (which is only known to the simulation) including labels for which agents has the visibility of each target. In Section 4, it is observed that targets enter or exit the cumulative FOV of the all agents. The lack of this information, instead of using the number of targets introduced for given setup, prevents the analyze of exact number of targets in the cumulative FOV of the all agents.

The scenarios introduced in Section 4 can be further diversify; unlimited FOV for agents, increased noise parameters, and increased ratio of false detections. Additional scenarios provide better insight on the model and ground its strengths and weaknesses (e. g., the threshold value of noise for the model, the performance of track generation in unlimited FOV).

Finally, a significant assumption posed by this thesis is the absence of communication and networking constraints. However, in physical implementations of such large swarm-based mobile sensor networks, issues such as communication noise and latencies pose an operationally relevant challenge. A number of potential avenues for future work include investigation of a hierarchical network, with node aggregation and approximation according to bandwidth, such as presented in [38]. For example, in [38], the authors observe that redundant data can be

ignored with aggregation techniques. Sensor information can also be clustered by assigning group leaders according to a pre-defined network organization or prioritization. Another candidate solution is a scalable sensing network for maintaining multi-target identity information, as examined by [39], where an identity mass-flow framework is used in order to decrease the computational workload of the system. Such methods for addressing scalability, robustness, throughput, and other network considerations offer significant potential for contribution.

---

## REFERENCES

---

- [1] J. Liu, M. Chu, and J. E. Reich, “Resource-Aware Multi-Target Tracking in Distributed Sensor Networks,” *Contract*, no. 1, pp. 1–23.
- [2] L. E. Parker, “Distributed Algorithms for Multi-Robot Observation of Multiple Moving Targets,” *Autonomous Robots*, vol. 12, pp. 231–255, May 2002.
- [3] L. B. Jean-Yves and V. Milan, “The Random Trip Model: Stability, Stationary, Regime and Perfect Simulation,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1153–1166, 2006.
- [4] A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, and S. Peyronnet, “Uniform Random Sampling of Traces in Very Large Models,” *Proceedings of the 1st international workshop on Random testing - RT '06*, p. 10, 2006.
- [5] J. Oudinet, “Uniform Random Walks in Very Large Models,” in *Second international workshop on random testing*, pp. 26–29, 2007.
- [6] T. Antal and S. Redner, “Escape of a Uniform Random Walk from an Interval,” *Journal of Statistical Physics*, vol. 123, pp. 1129–1144, July 2006.
- [7] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems 1,” *Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [8] P. Arambel and R. Mehra, “Estimation under unknown correlation: covariance intersection revisited,” *IEEE Transactions on Automatic Control*, vol. 47, pp. 1879–1882, Nov. 2002.
- [9] S. J. Julier and J. K. Uhlmann, “Using covariance intersection for SLAM,” *Robotics and Autonomous Systems*, vol. 55, pp. 3–20, Jan. 2007.
- [10] R. B. G. Wolfgang Niehsen, “Information Fusion based on Fast Covariance Intersection Filtering,” pp. 901–904.
- [11] A. Feldman, M. Hybinette, and T. Balch, “The multi-iterative closest point tracker: An on-line algorithm for tracking multiple interacting targets,” *Journal of Field Robotics*, pp. n/a–n/a, Jan. 2012.

- [12] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Selection of K in K-means clustering," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, pp. 103–119, Jan. 2005.
- [13] S. Salvador and P. Chan, "Determining the Number of Clusters / Segments in Hierarchical Clustering / Segmentation Algorithms," tech. rep., Florida Institute of Technology, Melbourne.
- [14] M. Liggins, I. Kadar, M. Alford, V. Vannicola, and S. Thomopoulos, "Distributed Fusion Architectures and Algorithms for Target Tracking," *Proceedings of the IEEE*, vol. 85, pp. 95–107, Jan. 1997.
- [15] C. K.C and Y.BAR-SHALOM, "Distributed Adaptive Estimation with Probabilistic Data Association," *Automatica*, vol. 25, no. 3, pp. 3259–369, 1989.
- [16] L. Chen, M. J. Wainwright, M. Cetin, and A. S. Willsky, "Multitarget-Multisensor Data Association Using the Tree-Reweighted Max-Product Algorithm," tech. rep., University of California, Berkeley, 2003.
- [17] L. Chen, M. J. Wainwright, M. Çetin, and A. S. Willsky, "Data association based on optimization in graphical models with application to sensor networks," *Mathematical and Computer Modelling*, vol. 43, pp. 1114–1135, May 2006.
- [18] A. G. Daronkolaei and V. Nazari, "A Joint Probability Data Association Filter Algorithm for Multiple Robot Tracking Problems," tech. rep., Amirkabir University of Technology, Tehran, Iran, Tehran, 2004.
- [19] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers, "People Tracking with Mobile Robots Using Sample-Based Joint Probabilistic Data Association Filters," *The International Journal of Robotics Research*, vol. 22, pp. 99–116, Feb. 2003.
- [20] A. Krause, J. Leskovec, and C. Guestrin, "Data association for topic intensity tracking," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 497–504, 2006.
- [21] R. J.A. and P. G.L., "Suboptimal Joint Probabilistic Data Association," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 2, pp. 510–517, 1993.

- [22] L. Pao and C. Frei, "A comparison of Parallel and Sequential Implementations of a Multisensor Multitarget Tracking Algorithm," in *Proceedings of 1995 American Control Conference - ACC'95*, vol. 3, pp. 1683–1687, American Autom Control Council, 1995.
- [23] W. R. A. and R. T. Rolf, "UAV Coordination for Autonomous Target Tracking," tech. rep., University of Washington, Seattle.
- [24] V. Jaco, G. S. J., and P. Patrick, "Monte Carlo Filtering for Multi-Target Tracking and Data Association," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, no. 1, pp. 309–332, 2006.
- [25] G. Pulford, "Taxonomy of multiple target tracking methods," *IEE Proceedings - Radar, Sonar and Navigation*, vol. 152, no. 5, p. 291, 2005.
- [26] X. R. LI and V. P. JILKOV, "Survey of Maneuvering Target Tracking. Part 1: Dynamic Models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, 2003.
- [27] L. Fan, Z. Wang, and H. Wang, "Multi-target Cell tracking based on classic kinetics," *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing Connecting the World Wirelessly - IWCMC '09*, p. 1340, 2009.
- [28] M. Mauricio F., *Agent-Based Simulation and Analyses of a Defensive UAV Swarm Against an Enemy UAV Swarm*. PhD thesis, Naval Postgraduate School, 2011.
- [29] Y. BAR-SHALOM, X. R. LI, and K. THIAGALINGAM, *Estimation with Applications to Tracking and Navigation*. New York, New York, USA: John Wiley & Sons, 2001.
- [30] D. Richard O., P. E. Hart, and D. G. Stork, *Pattern Classification*. second ed., 2000.
- [31] H. A. Edson and H. K. Karl, "Suboptimal JPDA for Tracking in the Presence of Clutter and Missed Detections," in *12th International Conference on Information Fusion*, (Seattle), pp. 818–825, 2009.
- [32] A. Bhattacharya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calculatta Mathematical Society*, vol. 35, pp. 99–109, 1943.
- [33] B. Road and P. Meer, "Real-Time Tracking of Non-Rigid Objects using Mean Shift 3 Bhattacharyya Coefficient Based Metric for Target Localization," *IEEE CVPR*, no. 8, 2000.

- [34] M. N. Hamedani, G. G. Tata, "On the determination of the bivariate normal distribution from distributions of linear combinations of the variables," *The American Mathematical Monthly*, vol. 82, no. 9, pp. 913–915, 1975.
- [35] N. Ushakov, "Probability Density Function," 2012.
- [36] M. J., "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.
- [37] R. Weil, F. Bourgeois, and J.-c. Lassalle, "An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices," *Communications of ACM*, vol. 14, no. 12, pp. 802–804, 1971.
- [38] D. Antonios, K. Yanniss, and R. Nick, "Dissemination of compressed historical information in sensor networks," in *VLDB*, pp. 439–461, 2006.
- [39] J. Shin, L. J. Guibas, and F. Zhao, "A Distributed Algorithm for Managing Multi-Target Identities in Wireless Ad-hoc Sensor Networks," *Lecture Notes in Computer Science*, pp. 1–16.

---

## Initial Distribution List

---

1. LTJG Umit Soylu, TUR Navy  
Arastirma Merkez Komutanligi  
Dogu Mah. Ankara Cad. No:252 P. K. 46 Pendik-Istanbul, Turkey
2. Dr. Timothy H. Chung  
Naval Postgraduate School  
Monterey, California
3. Dr. Joel Young  
Naval Postgraduate School  
Monterey, California
4. CAPT Jeffrey Kline, USN (ret)  
Director, Consortium for Robotics and Unmanned Systems Education and Research  
Naval Postgraduate School  
Monterey, California
5. Defense Technical Information Center  
Ft. Belvoir, Virginia
6. Dudley Knox Library  
Naval Postgraduate School  
Monterey, California
7. Marine Corps Representative  
Naval Postgraduate School  
Monterey, California
8. Directory, Training and Education, MCCDC, Code C46  
Quantico, Virginia
9. Marine Corps Tactical System Support Activity (Attn: Operations Officer)  
Camp Pendleton, California